Exhibit 9 to Complaint
Intellectual Ventures I LLC and Intellectual Ventures II LLC

Example American Count 2 Systems and Services
U.S. Patent No. 8,407,722 ("'722 Patent")

The Accused Systems and Services include without limitation American systems and services that utilize Kafka; all past, current, and future systems and services that operate in the same or substantially similar manner as the specifically identified systems and services; and all past, current, and future American systems and services that have the same or substantially similar features as the specifically identified systems and services ("Example American Count 2 Systems and Services" or "American Systems and Services").[1]

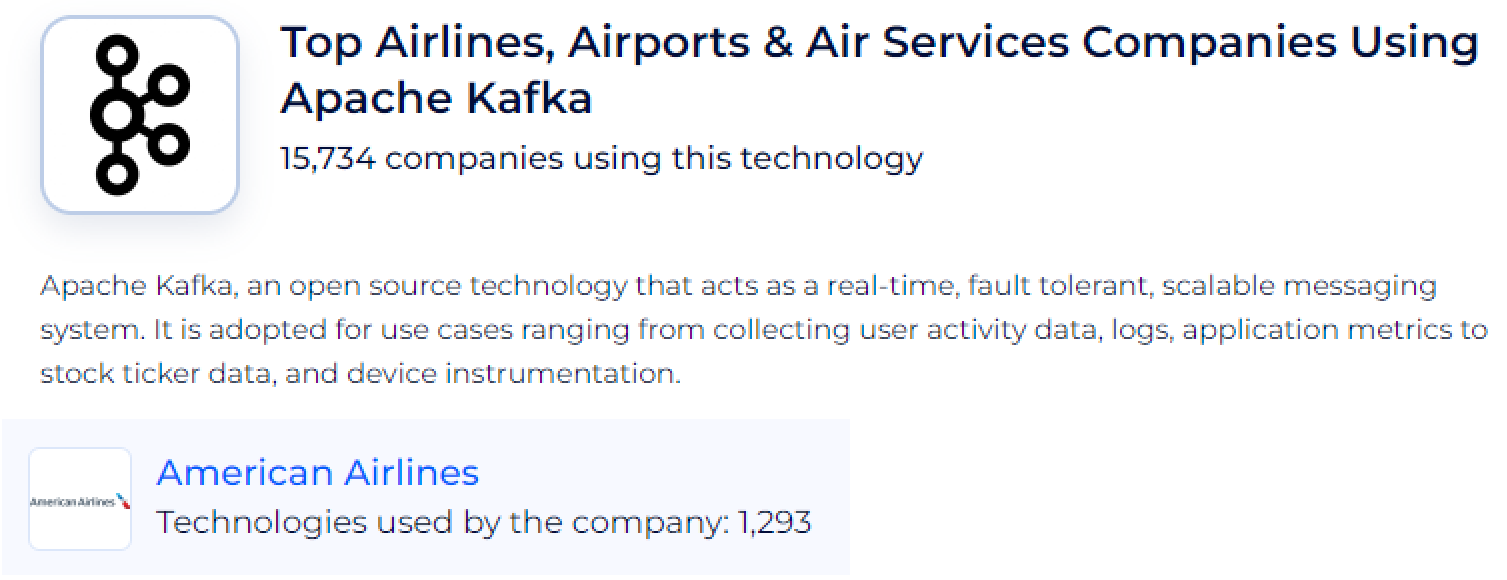
On information and belief, the American Systems and Services use Kafka in its private cloud(s). For example, American posts, or has posted, job opportunities that require familiarity with Kafka concepts.

- Example of a Senior Java Full Stack Developer at American Airlines whose position involves developing streaming and designing pipelines via Kafka. https://www.linkedin.com/in/rohitha-m6363/ (Last accessed on 9/19/24)
- Example of a Senior Data Engineer at American Airlines whose position involves developing streaming and designing pipelines via Kafka. https://www.linkedin.com/in/sriram1993/ (Last accessed on 9/19/24)
- Example of a Big Data Engineer at American Airlines whose position mentions Kafka. https://www.linkedin.com/in/madhu-sudhan-reddy-y-b3897b129/ (Last accessed on 9/19/24)
- Example of a Data Engineer at American Airlines whose position involves Kafka. https://www.linkedin.com/in/alisha-k-4b9571315.
- Example of a Data Engineer at American Airlines whose position involves real-time processing including Kafka. https://www.linkedin.com/in/sudheer-gajulapalli3.

As another example, American has announced cloud migration of legacy technology and efforts to modernize its mainframes and servers. Source: https://dxc.com/sg/en/insights/customer-stories/american-airlines-cloud-data-automation.

On information and belief, other information confirms American uses Kafka technology.

[1] For the avoidance of doubt, Plaintiffs do not accuse public clouds of American if those services are provided by a cloud provider with a license to Plaintiffs' patents that covers American's activities. IV will provide relevant license agreements for cloud providers in discovery. To the extent any of these licenses are relevant to American's activities, Plaintiffs will meet and confer with American about the impact of such license(s).



Top Airlines, Airports & Air Services Companies Using Apache Kafka

15,734 companies using this technology

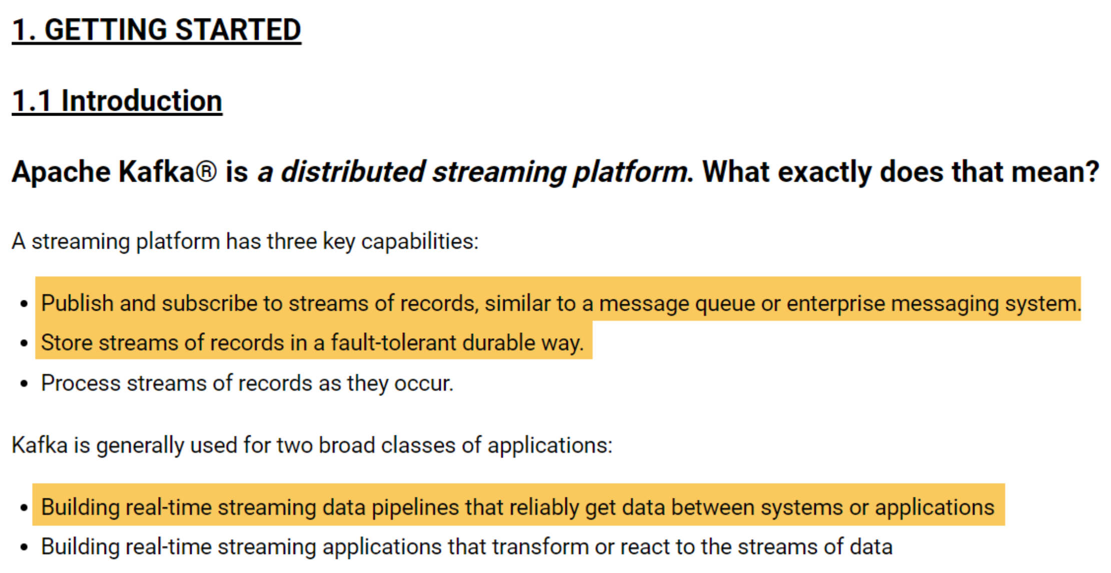
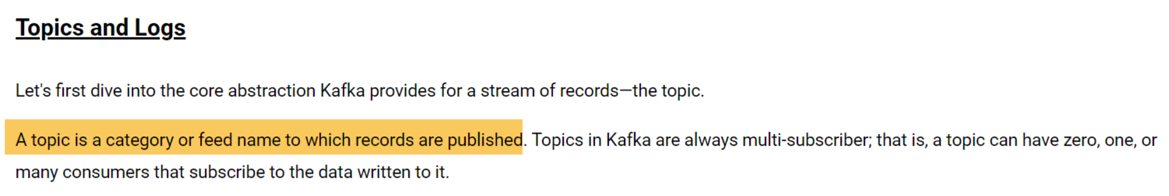
Apache Kafka, an open source technology that acts as a real-time, fault tolerant, scalable messaging system. It is adopted for use cases ranging from collecting user activity data, logs, application metrics to stock ticker data, and device instrumentation.

American Airlines

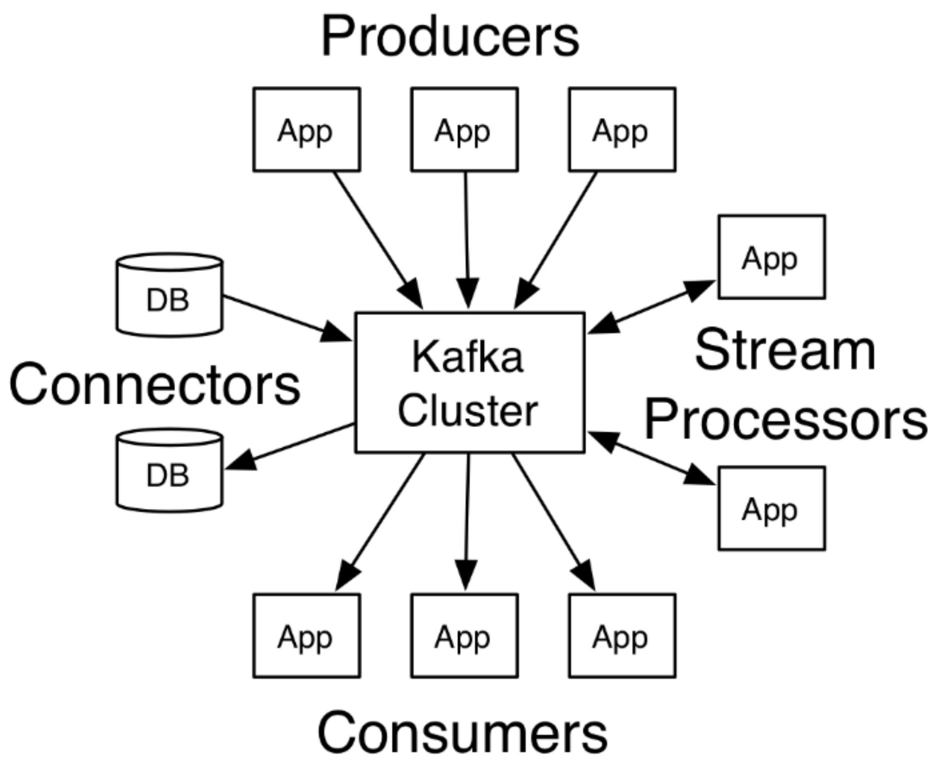
Technologies used by the company: 1,293

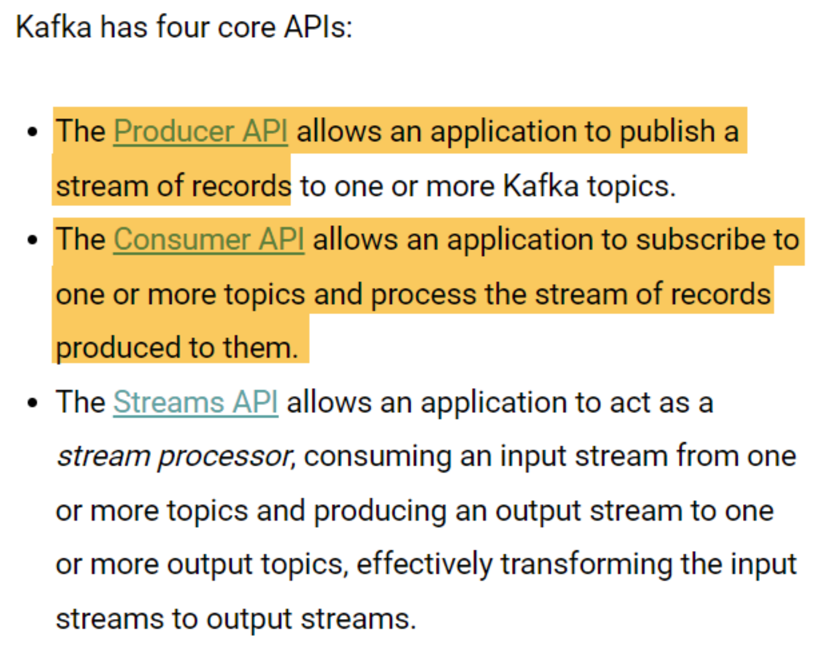
Source: https://www.zoominfo.com/tech/24100/apache-kafka-tech-from-transportation-airline-industry-by-revenue.[2]

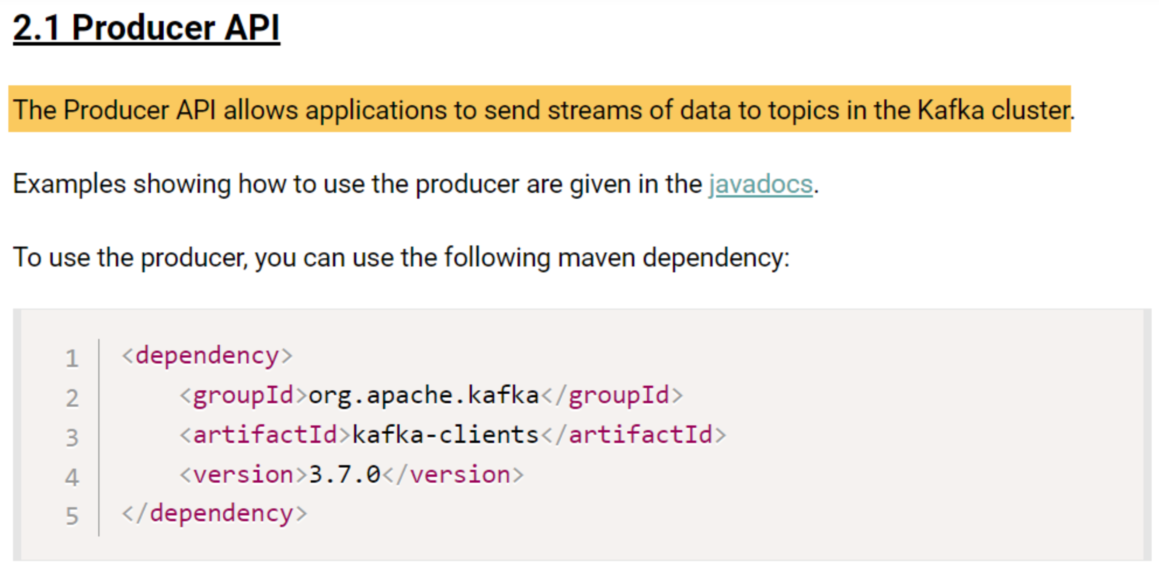
[2] All sources cited in this document were publicly accessible as of the filing date of the Complaint.

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
14. A method comprising:	To the extent this preamble is limiting, on information and belief, the American Count 2 Systems and Services practice a method. Kafka is a distributed streaming platform that enables publishing, subscribing, and storing streams of records.  **1. GETTING STARTED** **1.1 Introduction** **Apache Kafka® is *a distributed streaming platform*. What exactly does that mean?** A streaming platform has three key capabilities: • Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system. • Store streams of records in a fault-tolerant durable way. • Process streams of records as they occur. Kafka is generally used for two broad classes of applications: • Building real-time streaming data pipelines that reliably get data between systems or applications • Building real-time streaming applications that transform or react to the streams of data Source: https://kafka.apache.org/20/documentation.html.[3]  **Topics and Logs** Let's first dive into the core abstraction Kafka provides for a stream of records—the topic. A topic is a category or feed name to which records are published. Topics in Kafka are always multi-subscriber; that is, a topic can have zero, one, or many consumers that subscribe to the data written to it.

[3] Annotations added unless otherwise noted.

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	Source: https://kafka.apache.org/20/documentation.html.  Producers App App App DB Connectors DB Kafka Cluster App Stream Processors App App App App Consumers Source: https://kafka.apache.org/20/documentation.html.
14[a] providing, using a processing device of an input source, a data representation to a client device, different from the input source, coupled to a routing network, wherein the data	On information and belief, the American Count 2 Systems and Services practice providing, using a processing device of an input source, a data representation to a client device, different from the input source, coupled to a routing network, wherein the data representation includes at least one live object recognizable by the client device, and causing the client device to respond to the live object of the data representation by determining an object identifier (ID) of the live object and to register for updates of the live object with the routing network, such that registering the client device with the

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
representation includes at least one live object recognizable by the client device, and causing the client device to respond to the live object of the data representation by determining an object identifier (ID) of the live object and to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to a node in the routing network; and	routing network provides client connection information to a node in the routing network. Kafka includes a Producer API that allows applications to publish streams of records, and a Consumer API that allows applications to subscribe to one or more topics and process streams of records produced to them.  Kafka has four core APIs: • The Producer API allows an application to publish a stream of records to one or more Kafka topics. • The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them. • The Streams API allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams. Source: https://kafka.apache.org/20/documentation.html.

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 Source: https://kafka.apache.org/documentation.html.

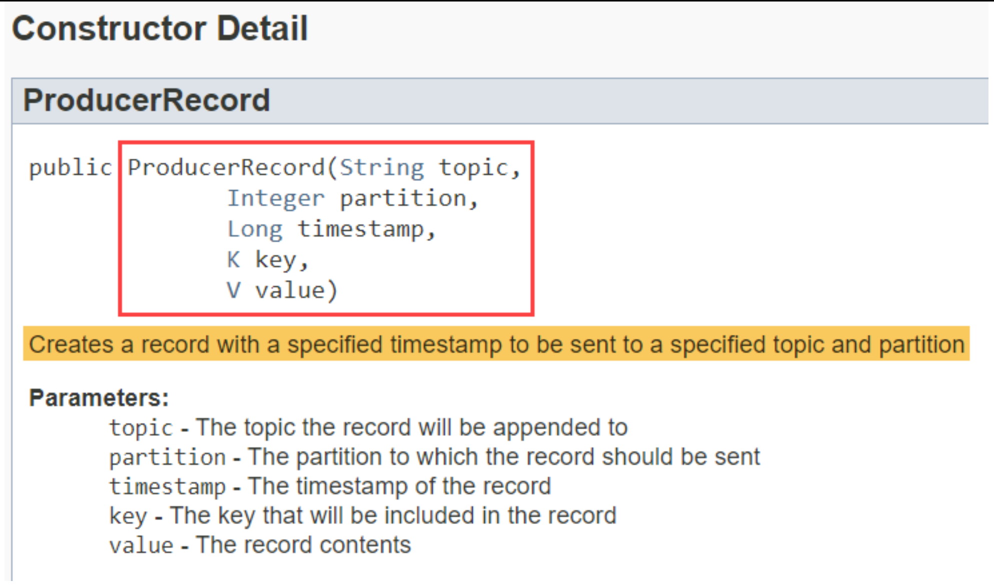
2.1 Producer API

The Producer API allows applications to send streams of data to topics in the Kafka cluster.

Examples showing how to use the producer are given in the javadocs.

To use the producer, you can use the following maven dependency:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>3.7.0</version>
</dependency>
```

<table>
<tr><th colspan="2">U.S. Patent No. 8,407,722 (Claim 14)</th></tr>
<tr><th>Claim(s)</th><th>Example American Count 2 Systems and Services</th></tr>
<tr><td></td><td>

Constructor Detail

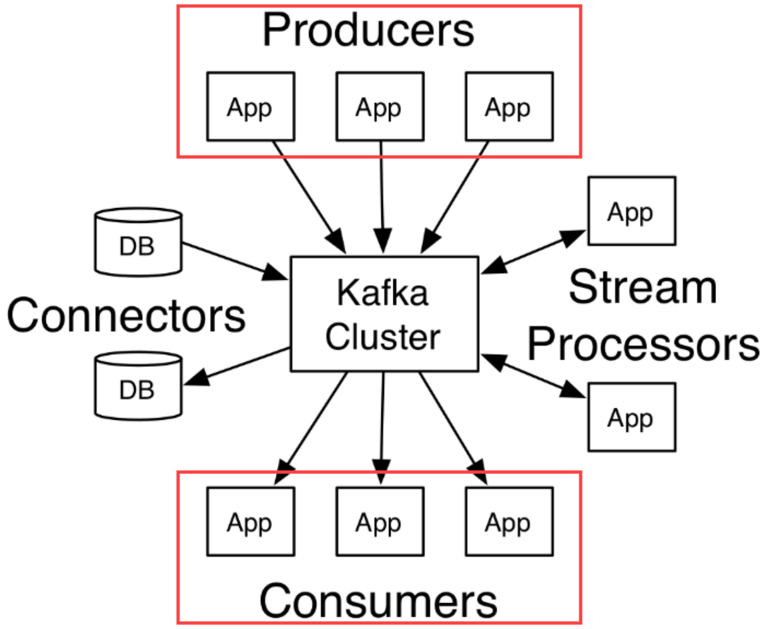
ProducerRecord

```
public ProducerRecord(String topic,
                      Integer partition,
                      Long timestamp,
                      K key,
                      V value)
```

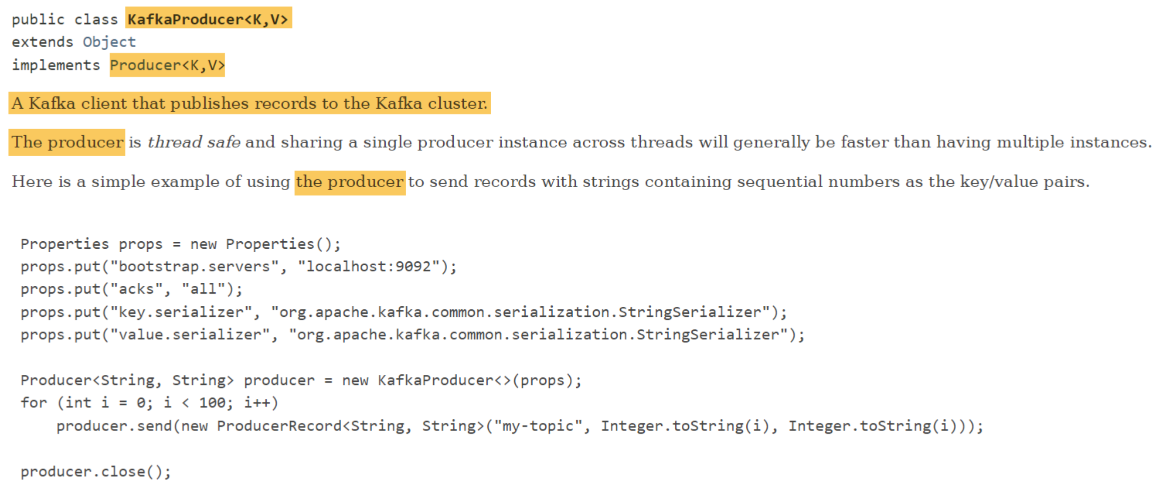
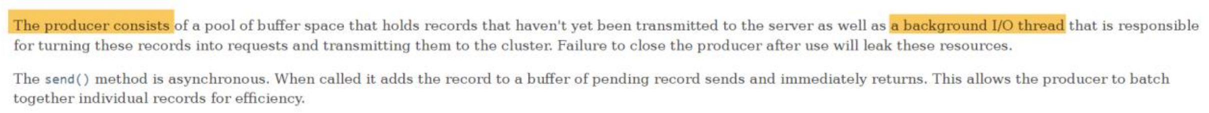
Creates a record with a specified timestamp to be sent to a specified topic and partition

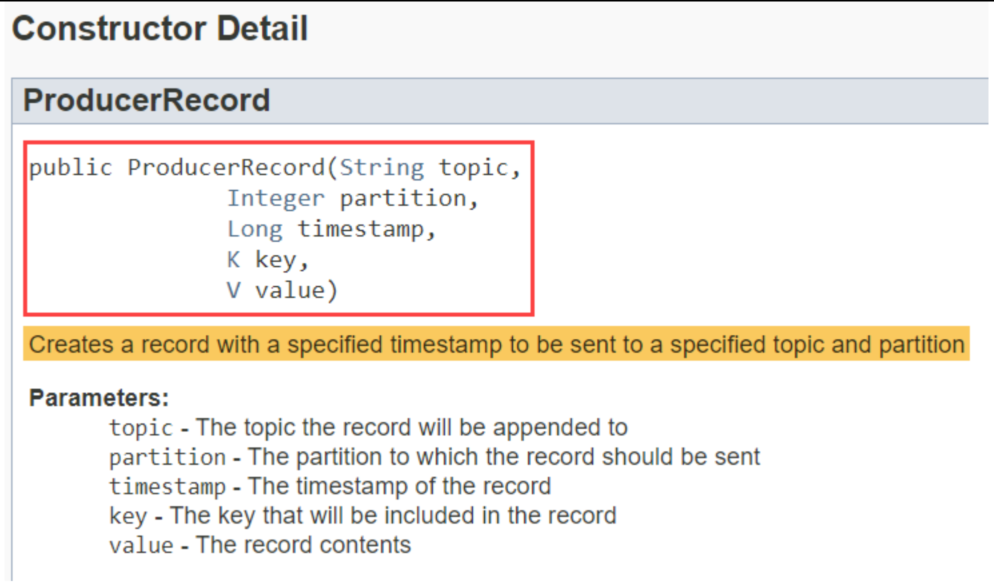
Parameters:
topic - The topic the record will be appended to
partition - The partition to which the record should be sent
timestamp - The timestamp of the record
key - The key that will be included in the record
value - The record contents

Source: https://kafka.apache.org/0102/javadoc/org/apache/kafka/clients/producer/ProducerRecord.html.</td></tr>
</table>

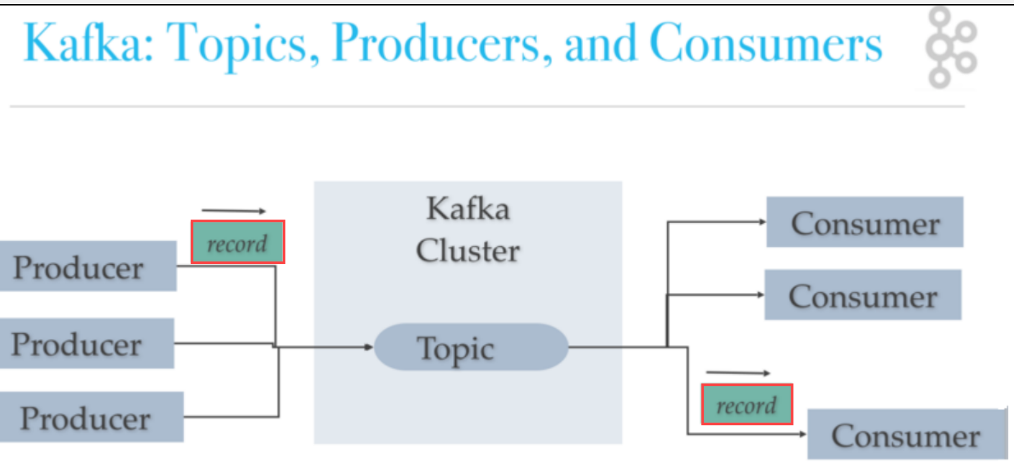
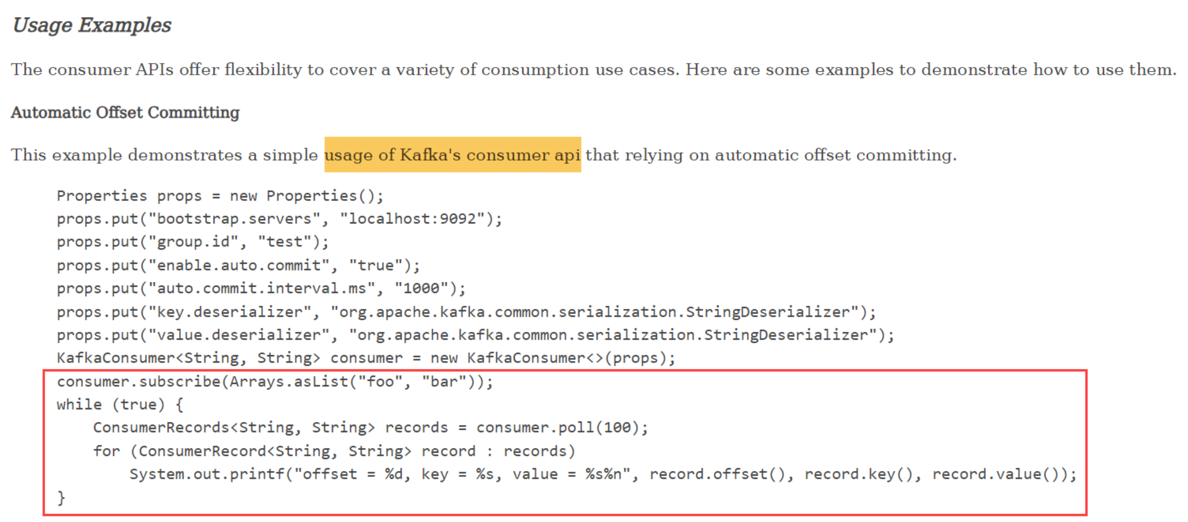
U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 Producers App App App DB Connectors DB Kafka Cluster App Stream Processors App App App App Consumers Source: https://kafka.apache.org/20/documentation.html. Kafka Producers publish records to a Kafka cluster.

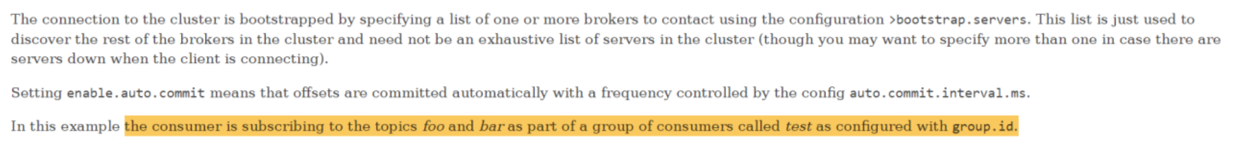
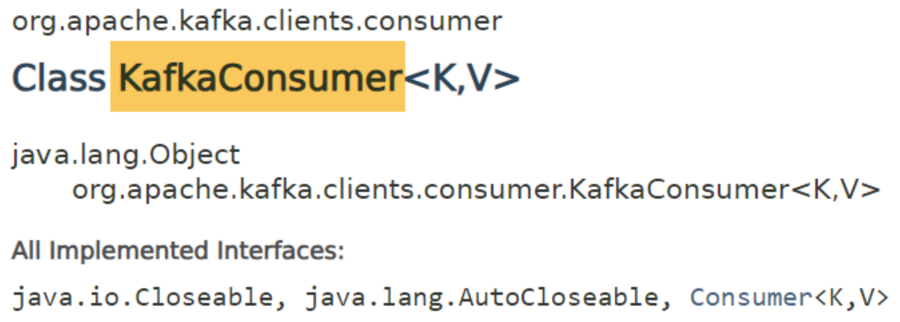
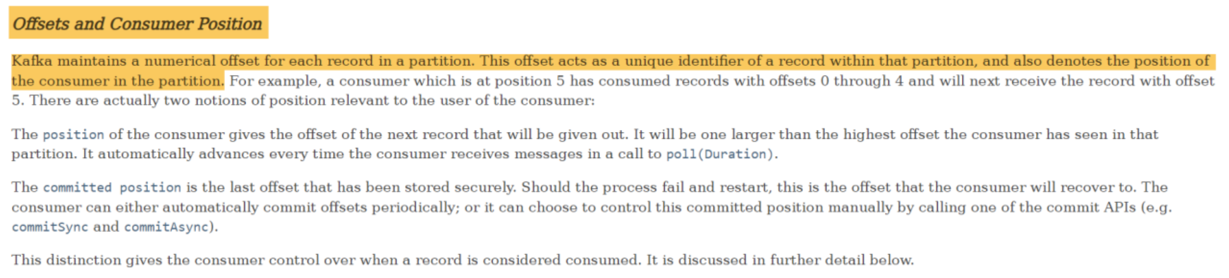
U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**

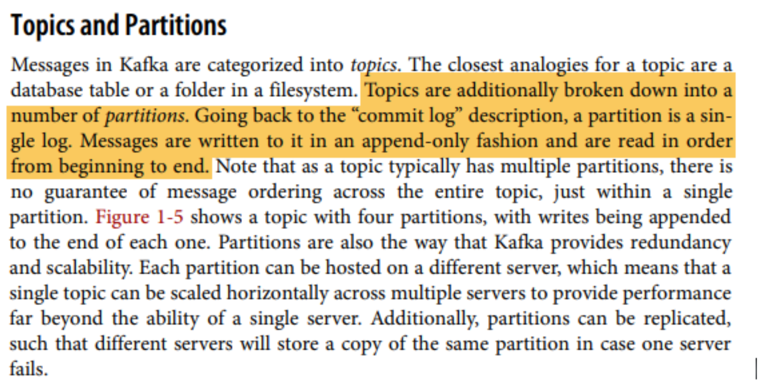
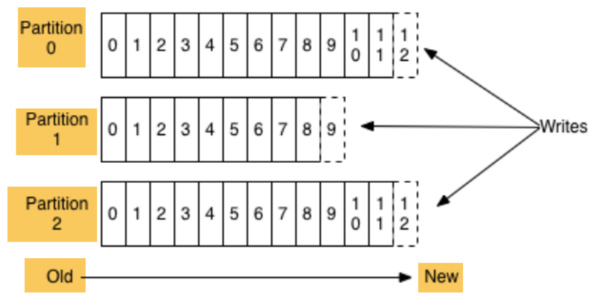
	
	`public class KafkaProducer<K,V>`
	`extends Object`
	`implements Producer<K,V>`
	A Kafka client that publishes records to the Kafka cluster.
	The producer is *thread safe* and sharing a single producer instance across threads will generally be faster than having multiple instances.
	Here is a simple example of using the producer to send records with strings containing sequential numbers as the key/value pairs.
	`Properties props = new Properties();`
	`props.put("bootstrap.servers", "localhost:9092");`
	`props.put("acks", "all");`
	`props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");`
	`props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");`
	
	`Producer<String, String> producer = new KafkaProducer<>(props);`
	`for (int i = 0; i < 100; i++)`
	`    producer.send(new ProducerRecord<String, String>("my-topic", Integer.toString(i), Integer.toString(i)));`
	
	`producer.close();`
	
	Source: https://kafka.apache.org/23/javadoc/org/apache/kafka/clients/producer/KafkaProducer.html.
	
	The producer consists of a pool of buffer space that holds records that haven't yet been transmitted to the server as well as a background I/O thread that is responsible for turning these records into requests and transmitting them to the cluster. Failure to close the producer after use will leak these resources.
	The `send()` method is asynchronous. When called it adds the record to a buffer of pending record sends and immediately returns. This allows the producer to batch together individual records for efficiency.
	
	Source: https://kafka.apache.org/23/javadoc/org/apache/kafka/clients/producer/KafkaProducer.html.
	Kafka Producers create ProducerRecords that are sent to a specific topic and partition. These records can be received and processed by Consumers.

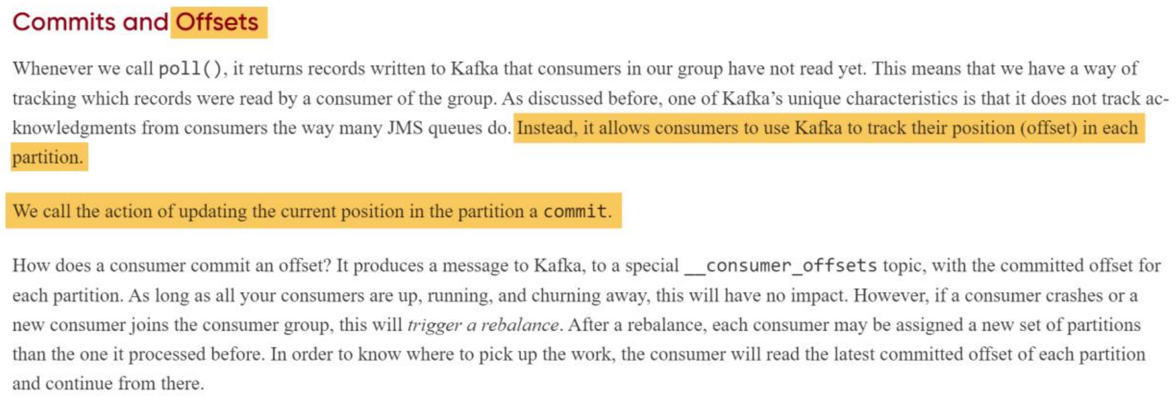
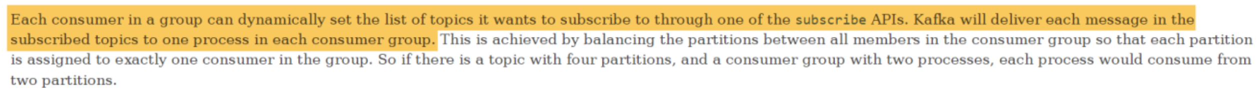
U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 Constructor Detail ProducerRecord `public ProducerRecord(String topic, Integer partition, Long timestamp, K key, V value)` Creates a record with a specified timestamp to be sent to a specified topic and partition **Parameters:** `topic` - The topic the record will be appended to `partition` - The partition to which the record should be sent `timestamp` - The timestamp of the record `key` - The key that will be included in the record `value` - The record contents Source: https://kafka.apache.org/0102/javadoc/org/apache/kafka/clients/producer/ProducerRecord.html.

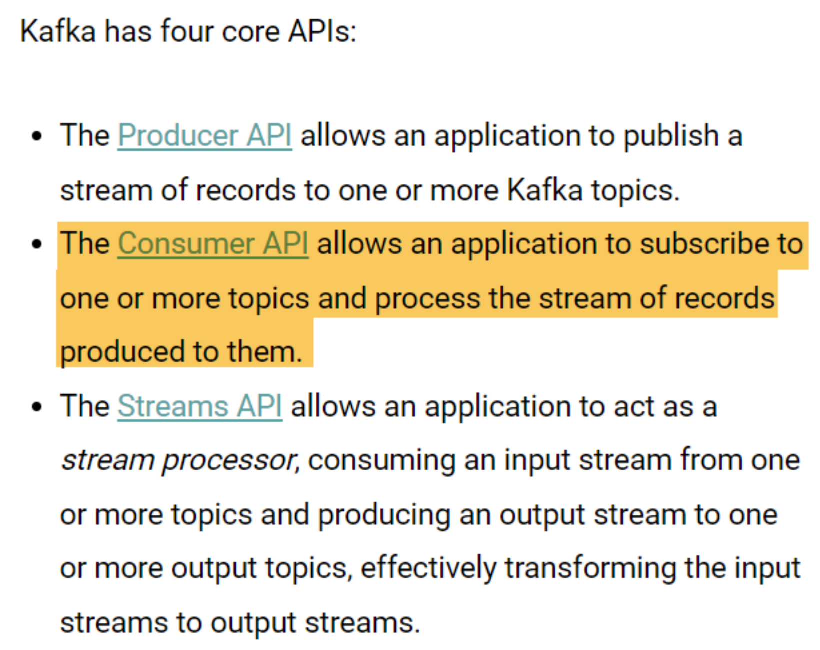
U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**

	 Kafka: Topics, Producers, and Consumers
	Producer
	Producer
	Producer
	record
	Kafka Cluster
	Topic
	Consumer
	Consumer
	record
	Consumer Source: http://cloudurable.com/blog/kafka-architecture/index.html.  *Usage Examples* The consumer APIs offer flexibility to cover a variety of consumption use cases. Here are some examples to demonstrate how to use them. **Automatic Offset Committing** This example demonstrates a simple usage of Kafka's consumer api that relying on automatic offset committing.
	`Properties props = new Properties();`
	`props.put("bootstrap.servers", "localhost:9092");`
	`props.put("group.id", "test");`
	`props.put("enable.auto.commit", "true");`
	`props.put("auto.commit.interval.ms", "1000");`
	`props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");`
	`props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");`
	`KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);`
	`consumer.subscribe(Arrays.asList("foo", "bar"));`
	`while (true) {`
	`    ConsumerRecords<String, String> records = consumer.poll(100);`
	`    for (ConsumerRecord<String, String> record : records)`
	`        System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(), record.value());`
	`}`
	Source: https://kafka.apache.org/10/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html.

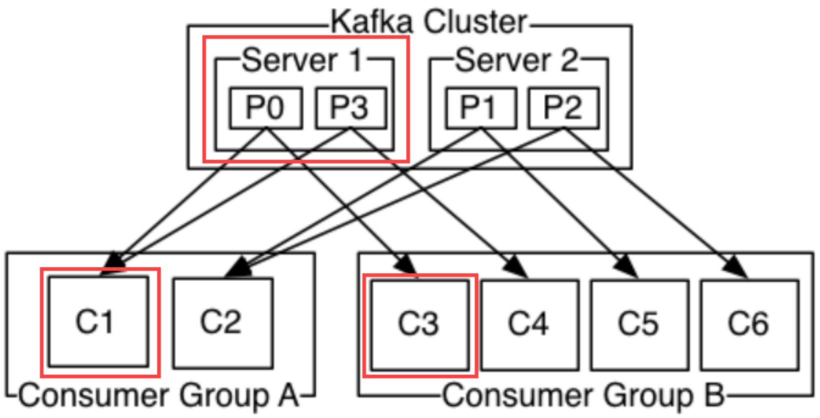
U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 The connection to the cluster is bootstrapped by specifying a list of one or more brokers to contact using the configuration >`bootstrap.servers`. This list is just used to discover the rest of the brokers in the cluster and need not be an exhaustive list of servers in the cluster (though you may want to specify more than one in case there are servers down when the client is connecting). Setting `enable.auto.commit` means that offsets are committed automatically with a frequency controlled by the config `auto.commit.interval.ms`. In this example the consumer is subscribing to the topics *foo* and *bar* as part of a group of consumers called *test* as configured with `group.id`. Source: https://kafka.apache.org/10/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html . Kafka topics are divided into partitions, and Kafka uses offsets that act uas an identifier of a record within a partition and denotes the position of the consumer to the partition.  org.apache.kafka.clients.consumer Class KafkaConsumer<K,V> java.lang.Object org.apache.kafka.clients.consumer.KafkaConsumer<K,V> All Implemented Interfaces: `java.io.Closeable, java.lang.AutoCloseable, Consumer<K,V>` Source: https://kafka.apache.org/22/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html.  *Offsets and Consumer Position* Kafka maintains a numerical offset for each record in a partition. This offset acts as a unique identifier of a record within that partition, and also denotes the position of the consumer in the partition. For example, a consumer which is at position 5 has consumed records with offsets 0 through 4 and will next receive the record with offset 5. There are actually two notions of position relevant to the user of the consumer: The `position` of the consumer gives the offset of the next record that will be given out. It will be one larger than the highest offset the consumer has seen in that partition. It automatically advances every time the consumer receives messages in a call to `poll(Duration)`. The `committed position` is the last offset that has been stored securely. Should the process fail and restart, this is the offset that the consumer will recover to. The consumer can either automatically commit offsets periodically; or it can choose to control this committed position manually by calling one of the commit APIs (e.g. `commitSync` and `commitAsync`). This distinction gives the consumer control over when a record is considered consumed. It is discussed in further detail below. Source: https://kafka.apache.org/22/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html.

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 **Topics and Partitions** Messages in Kafka are categorized into *topics*. The closest analogies for a topic are a database table or a folder in a filesystem. Topics are additionally broken down into a number of *partitions*. Going back to the "commit log" description, a partition is a single log. Messages are written to it in an append-only fashion and are read in order from beginning to end. Note that as a topic typically has multiple partitions, there is no guarantee of message ordering across the entire topic, just within a single partition. Figure 1-5 shows a topic with four partitions, with writes being appended to the end of each one. Partitions are also the way that Kafka provides redundancy and scalability. Each partition can be hosted on a different server, which means that a single topic can be scaled horizontally across multiple servers to provide performance far beyond the ability of a single server. Additionally, partitions can be replicated, such that different servers will store a copy of the same partition in case one server fails. Source: Kafka: The Definitive Guide (2nd Ed.), O'Reilly Publishing (2021). For each topic, the Kafka cluster maintains a partitioned log that looks like this: **Anatomy of a Topic**  Partition 0 0 1 2 3 4 5 6 7 8 9 10 11 12 Partition 1 0 1 2 3 4 5 6 7 8 9 Partition 2 0 1 2 3 4 5 6 7 8 9 10 11 12 Writes Old New Source: https://kafka.apache.org/20/documentation.html.

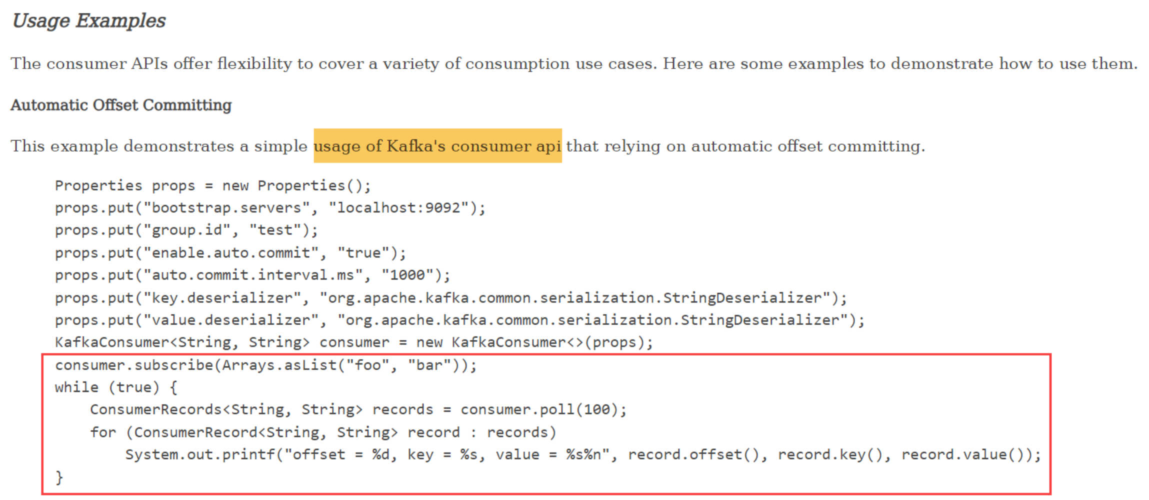
U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 Commits and Offsets Whenever we call `poll()`, it returns records written to Kafka that consumers in our group have not read yet. This means that we have a way of tracking which records were read by a consumer of the group. As discussed before, one of Kafka's unique characteristics is that it does not track acknowledgments from consumers the way many JMS queues do. Instead, it allows consumers to use Kafka to track their position (offset) in each partition. We call the action of updating the current position in the partition a `commit`. How does a consumer commit an offset? It produces a message to Kafka, to a special `__consumer_offsets` topic, with the committed offset for each partition. As long as all your consumers are up, running, and churning away, this will have no impact. However, if a consumer crashes or a new consumer joins the consumer group, this will *trigger a rebalance*. After a rebalance, each consumer may be assigned a new set of partitions than the one it processed before. In order to know where to pick up the work, the consumer will read the latest committed offset of each partition and continue from there. Source: https://www.oreilly.com/library/view/kafka-the-definitive/9781491936153/ch04.html. Kafka Consumers can identify topics that it wants to subscribe to through subscribe APIs and receive records through offset processing.  Each consumer in a group can dynamically set the list of topics it wants to subscribe to through one of the `subscribe` APIs. Kafka will deliver each message in the subscribed topics to one process in each consumer group. This is achieved by balancing the partitions between all members in the consumer group so that each partition is assigned to exactly one consumer in the group. So if there is a topic with four partitions, and a consumer group with two processes, each process would consume from two partitions. Source: https://kafka.apache.org/22/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html.

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 Kafka has four core APIs: • The Producer API allows an application to publish a stream of records to one or more Kafka topics. • The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them. • The Streams API allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams. Source: https://kafka.apache.org/20/documentation.html.

<table>
<tr><th colspan="2">U.S. Patent No. 8,407,722 (Claim 14)</th></tr>
<tr><th>Claim(s)</th><th>Example American Count 2 Systems and Services</th></tr>
<tr><td></td><td>



Source: https://kafka.apache.org/20/documentation.html.



Usage Examples

The consumer APIs offer flexibility to cover a variety of consumption use cases. Here are some examples to demonstrate how to use them.

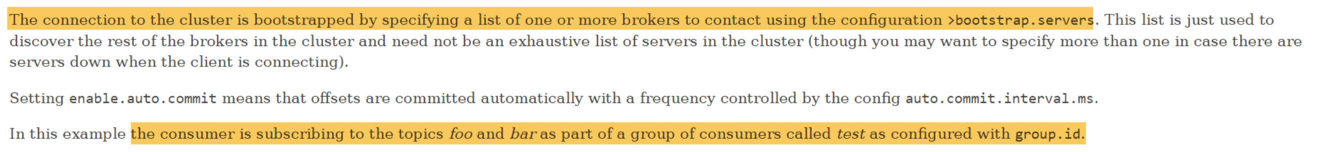
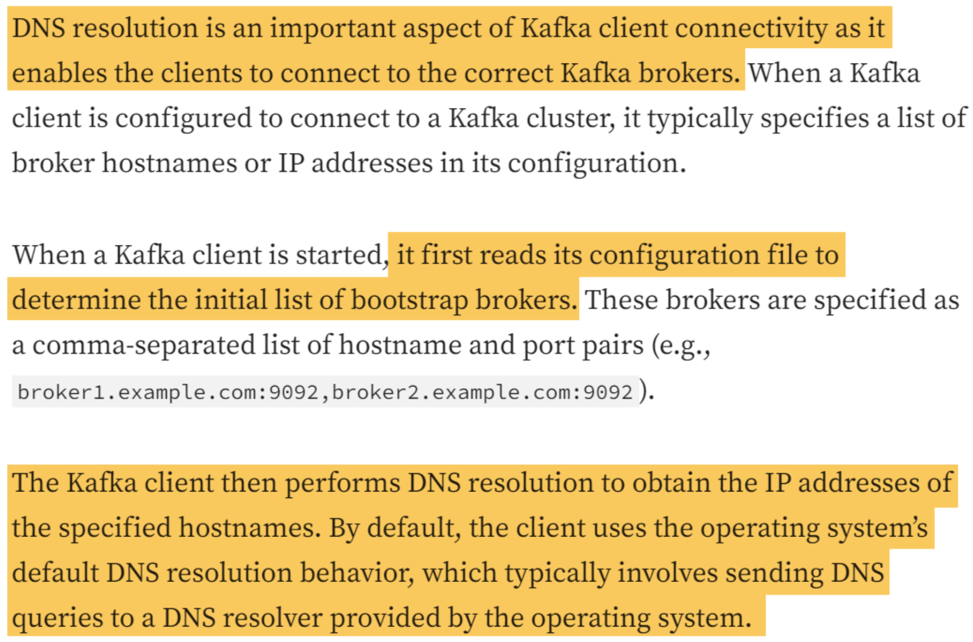
Automatic Offset Committing

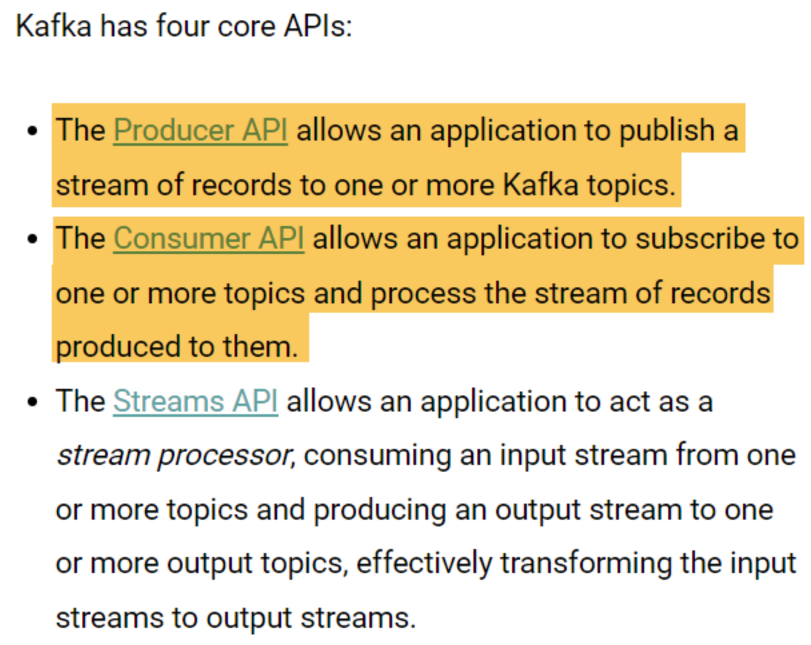
This example demonstrates a simple usage of Kafka's consumer api that relying on automatic offset committing.

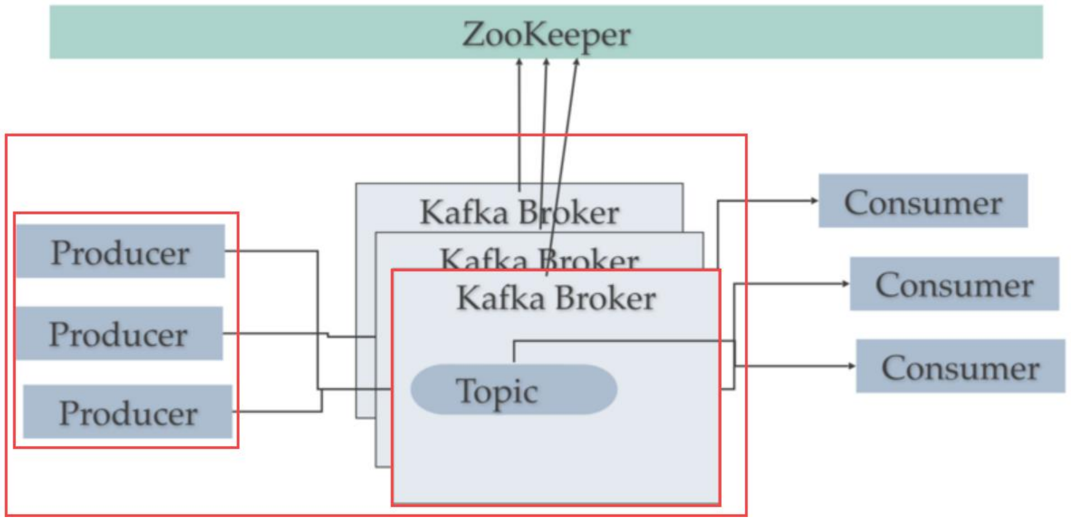
```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("group.id", "test");
props.put("enable.auto.commit", "true");
props.put("auto.commit.interval.ms", "1000");
props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("foo", "bar"));
while (true) {
    ConsumerRecords<String, String> records = consumer.poll(100);
    for (ConsumerRecord<String, String> record : records)
        System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(), record.value());
}
```

Source: https://kafka.apache.org/10/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html.

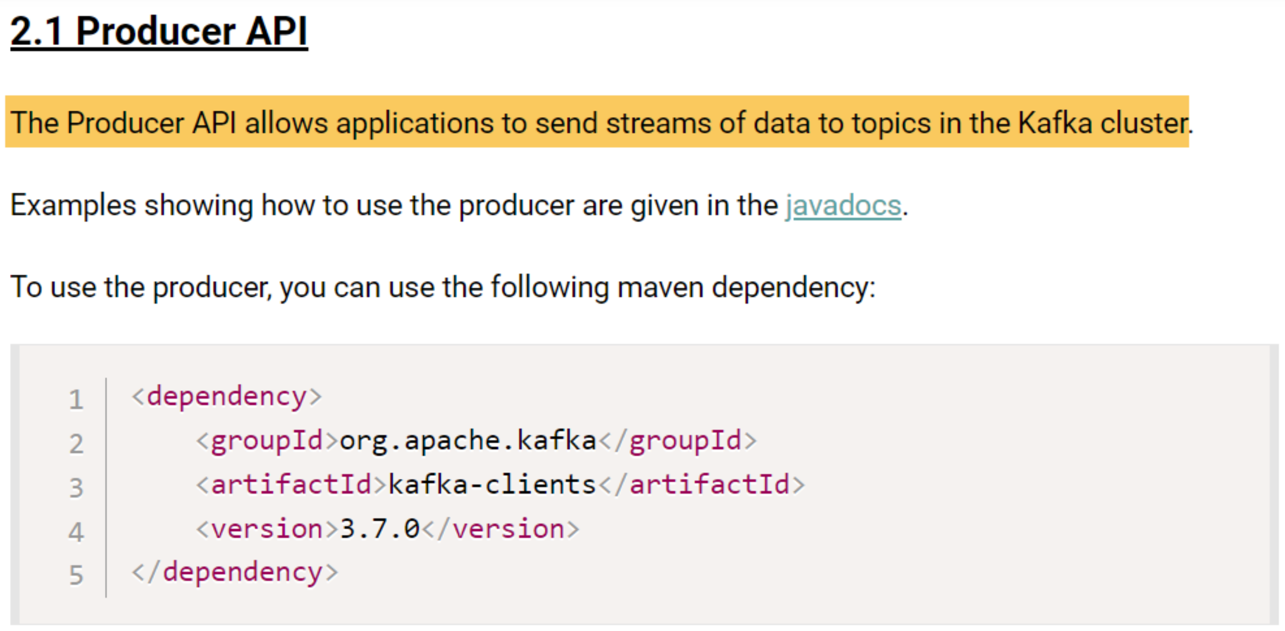
</td></tr>
</table>

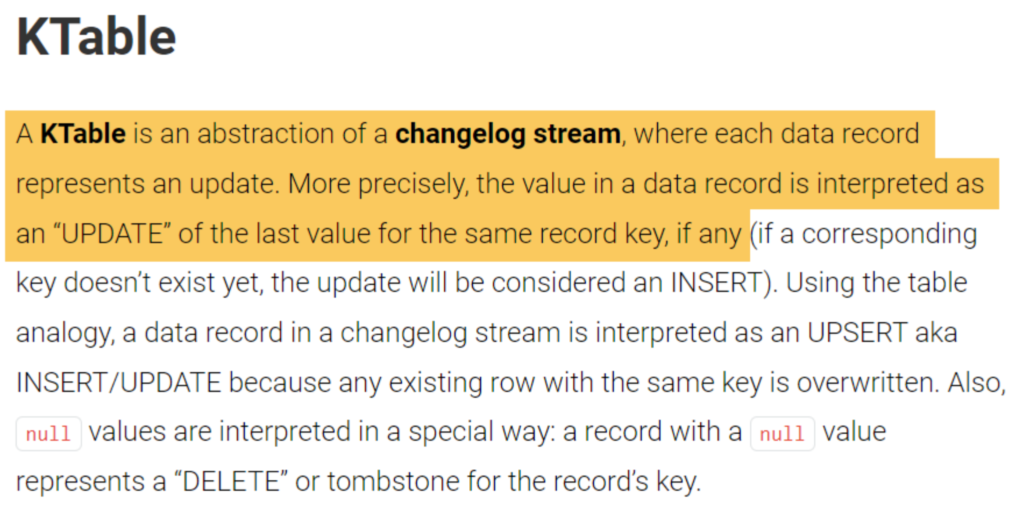
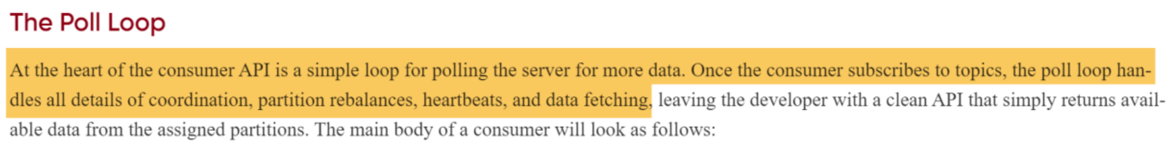
U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 The connection to the cluster is bootstrapped by specifying a list of one or more brokers to contact using the configuration `>bootstrap.servers`. This list is just used to discover the rest of the brokers in the cluster and need not be an exhaustive list of servers in the cluster (though you may want to specify more than one in case there are servers down when the client is connecting). Setting `enable.auto.commit` means that offsets are committed automatically with a frequency controlled by the config `auto.commit.interval.ms`. In this example the consumer is subscribing to the topics *foo* and *bar* as part of a group of consumers called *test* as configured with `group.id`. Source: https://kafka.apache.org/10/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html. Kafka uses DNS resolution for Kafka client connectivity to Kafka brokers.  DNS resolution is an important aspect of Kafka client connectivity as it enables the clients to connect to the correct Kafka brokers. When a Kafka client is configured to connect to a Kafka cluster, it typically specifies a list of broker hostnames or IP addresses in its configuration. When a Kafka client is started, it first reads its configuration file to determine the initial list of bootstrap brokers. These brokers are specified as a comma-separated list of hostname and port pairs (e.g., `broker1.example.com:9092,broker2.example.com:9092`). The Kafka client then performs DNS resolution to obtain the IP addresses of the specified hostnames. By default, the client uses the operating system's default DNS resolution behavior, which typically involves sending DNS queries to a DNS resolver provided by the operating system. Source: https://levelup.gitconnected.com/dns-resolution-for-kafka-clients-2e76b28e4717.
14[b] sending, using the processing device of the input source, an update	On information and belief, the American Count 2 Systems and Services practice sending, using the processing device of the input source, an update message to the routing network, wherein the update message identifies the live object and contains update data that updates a property of the live object.

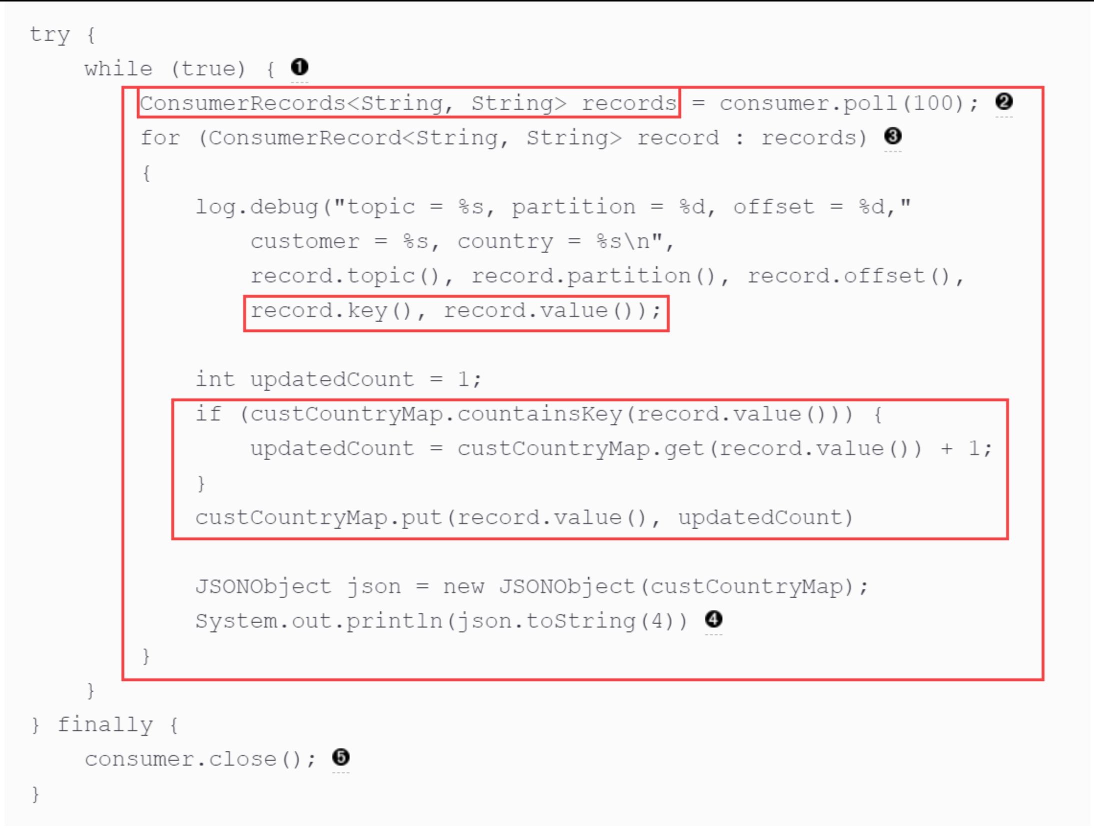
U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
message to the routing network, wherein the update message identifies the live object and contains update data that updates a property of the live object,	Kafka Producers can use the Producer API to send streams of data to topics in a Kafka cluster.  Kafka has four core APIs: • The Producer API allows an application to publish a stream of records to one or more Kafka topics. • The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them. • The Streams API allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams. Source: https://kafka.apache.org/20/documentation.html.

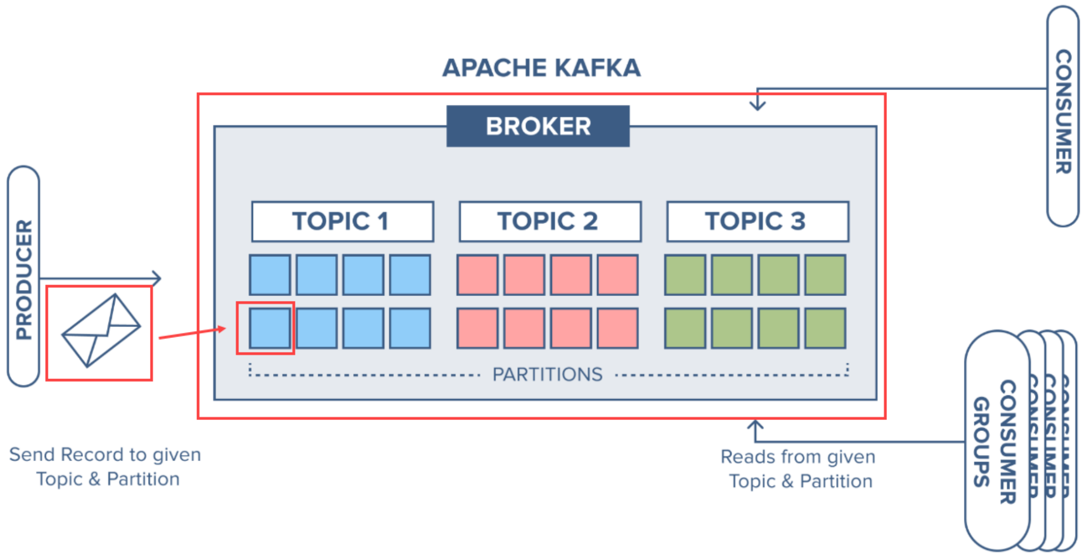
U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 ZooKeeper Producer Producer Producer Kafka Broker Kafka Broker Kafka Broker Topic Consumer Consumer Consumer Source: http://cloudurable.com/blog/kafka-architecture/index.html.

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**

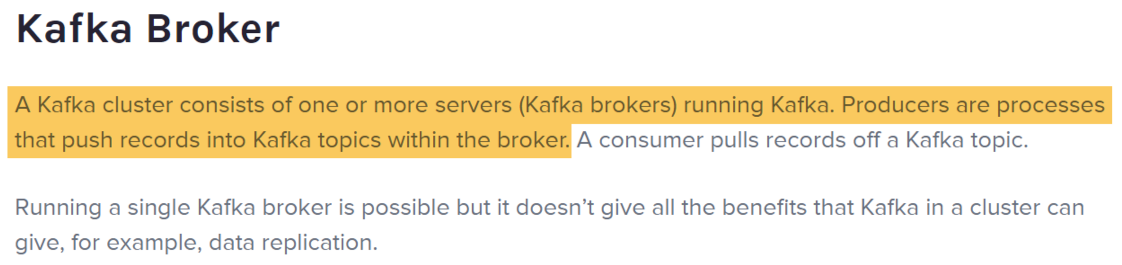
	 **2.1 Producer API**
	
	 The Producer API allows applications to send streams of data to topics in the Kafka cluster.
	
	 Examples showing how to use the producer are given in the javadocs.
	
	 To use the producer, you can use the following maven dependency:
	
	 `<dependency>`
	 `    <groupId>org.apache.kafka</groupId>`
	 `    <artifactId>kafka-clients</artifactId>`
	 `    <version>3.7.0</version>`
	 `</dependency>`
	
	 Source: https://kafka.apache.org/documentation.html.

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 **KTable** A **KTable** is an abstraction of a **changelog stream**, where each data record represents an update. More precisely, the value in a data record is interpreted as an "UPDATE" of the last value for the same record key, if any (if a corresponding key doesn't exist yet, the update will be considered an INSERT). Using the table analogy, a data record in a changelog stream is interpreted as an UPSERT aka INSERT/UPDATE because any existing row with the same key is overwritten. Also, `null` values are interpreted in a special way: a record with a `null` value represents a "DELETE" or tombstone for the record's key. Source: https://docs.confluent.io/platform/current/streams/concepts.html.  The Poll Loop At the heart of the consumer API is a simple loop for polling the server for more data. Once the consumer subscribes to topics, the poll loop handles all details of coordination, partition rebalances, heartbeats, and data fetching, leaving the developer with a clean API that simply returns available data from the assigned partitions. The main body of a consumer will look as follows: Source: https://www.oreilly.com/library/view/kafka-the-definitive/9781491936153/ch04.html.

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 `try {` `while (true) { ❶` `ConsumerRecords<String, String> records = consumer.poll(100); ❷` `for (ConsumerRecord<String, String> record : records) ❸` `{` `log.debug("topic = %s, partition = %d, offset = %d,"` `customer = %s, country = %s\n",` `record.topic(), record.partition(), record.offset(),` `record.key(), record.value());` `int updatedCount = 1;` `if (custCountryMap.countainsKey(record.value())) {` `updatedCount = custCountryMap.get(record.value()) + 1;` `}` `custCountryMap.put(record.value(), updatedCount)` `JSONObject json = new JSONObject(custCountryMap);` `System.out.println(json.toString(4)) ❹` `}` `}` `} finally {` `consumer.close(); ❺` `}` Source: https://www.oreilly.com/library/view/kafka-the-definitive/9781491936153/ch04.html.
14[c] wherein a gateway device at the routing network is configured to identify a category of the update message based on the input source, to determine a node type to which the identified	On information and belief, the American Count 2 Systems and Services practice a gateway device at the routing network is configured to identify a category of the update message based on the input source, to determine a node type to which the identified category maps, and to route the update message to the node, having the node type, at the routing network. Kafka clusters include Kafka brokers, where Kafka Producers push records to Kafka topics via a broker. Kafka Consumers pull records off a Kafka topic.

<table>
<tr><th colspan="2">U.S. Patent No. 8,407,722 (Claim 14)</th></tr>
<tr><th>Claim(s)</th><th>Example American Count 2 Systems and Services</th></tr>
<tr><td>category maps, and to route the update message to the node, having the node type, at the routing network,</td><td>

Source: https://www.cloudkarafka.com/blog/part1-kafka-for-beginners-what-is-apache-kafka.html.

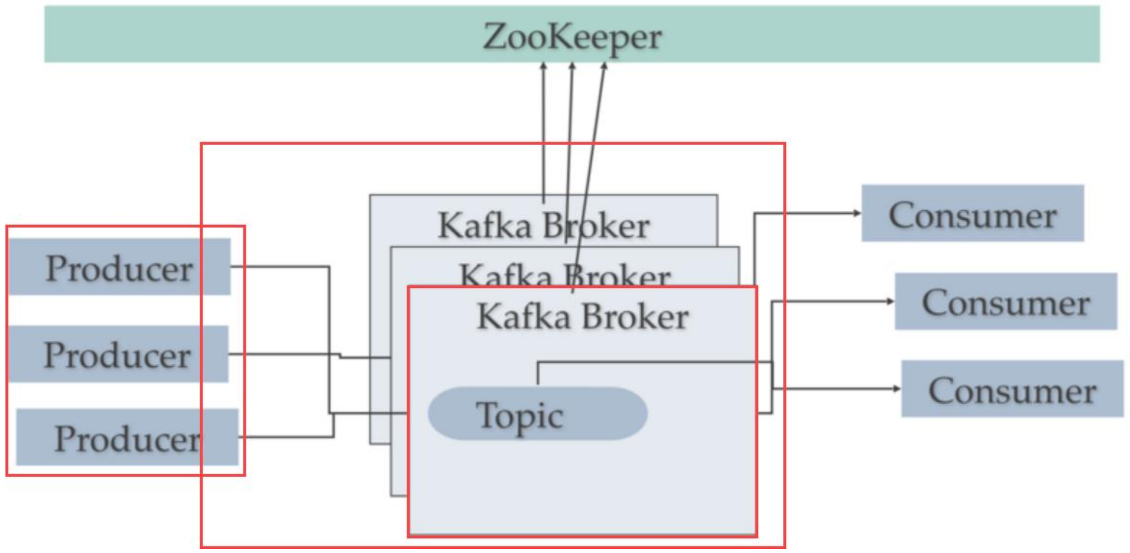
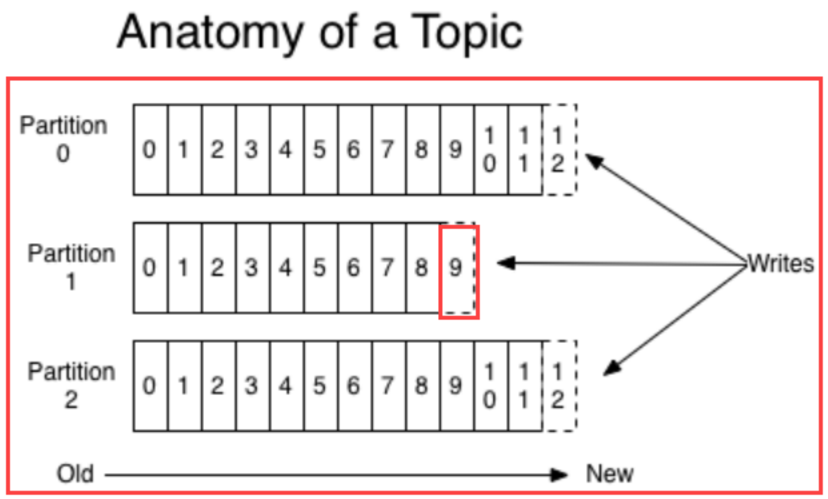


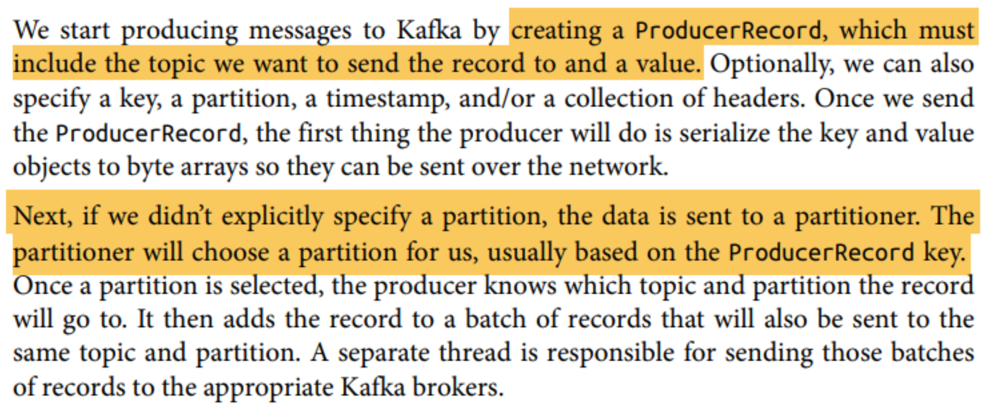
Kafka Broker

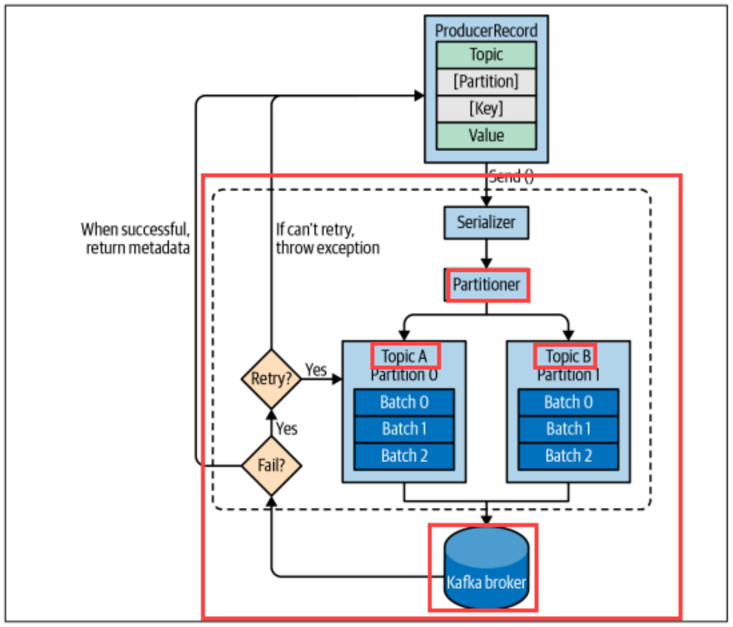
A Kafka cluster consists of one or more servers (Kafka brokers) running Kafka. Producers are processes that push records into Kafka topics within the broker. A consumer pulls records off a Kafka topic.

Running a single Kafka broker is possible but it doesn't give all the benefits that Kafka in a cluster can give, for example, data replication.

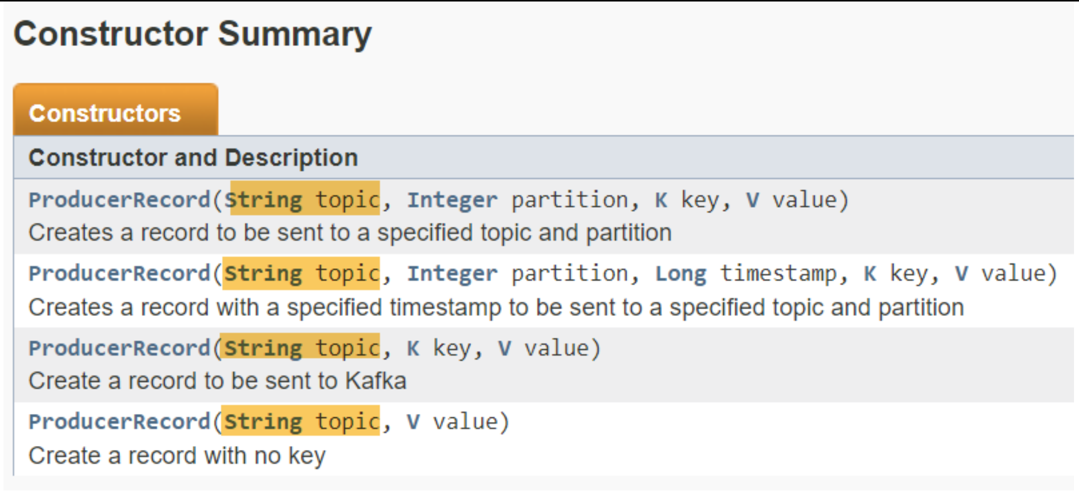
Source: https://www.cloudkarafka.com/blog/part1-kafka-for-beginners-what-is-apache-kafka.html.</td></tr>
</table>

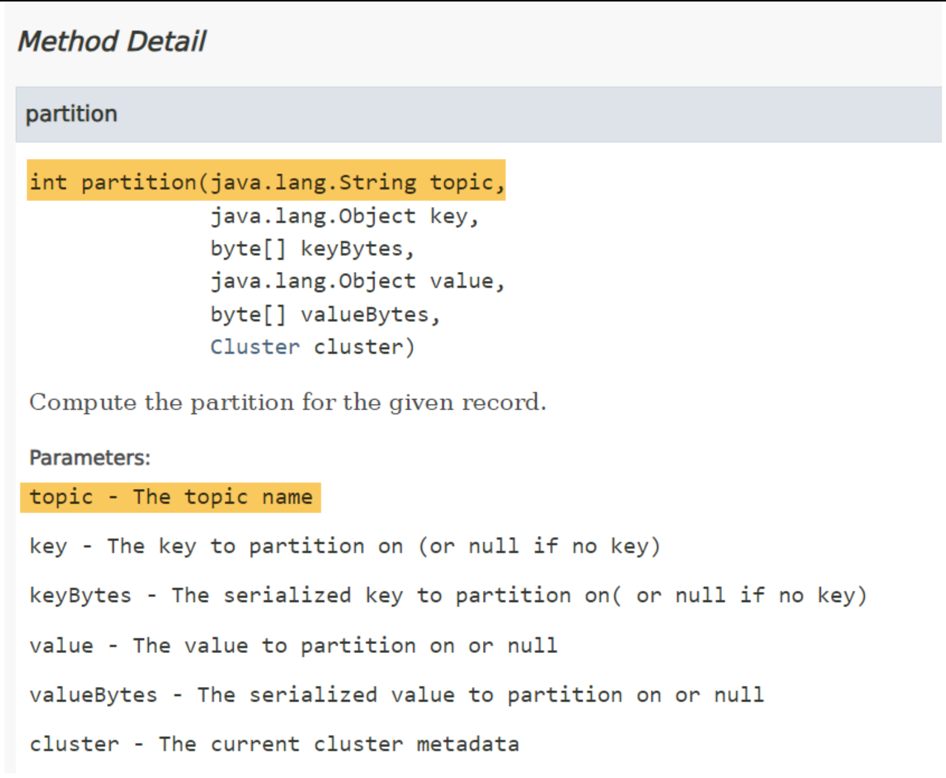
U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 ZooKeeper Producer Producer Producer Kafka Broker Kafka Broker Kafka Broker Topic Consumer Consumer Consumer Source: http://cloudurable.com/blog/kafka-architecture/index.html.  Anatomy of a Topic Partition 0 Partition 1 Partition 2 Writes Old New

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	Source: https://kafka.apache.org/20/documentation.html.  We start producing messages to Kafka by creating a `ProducerRecord`, which must include the topic we want to send the record to and a value. Optionally, we can also specify a key, a partition, a timestamp, and/or a collection of headers. Once we send the `ProducerRecord`, the first thing the producer will do is serialize the key and value objects to byte arrays so they can be sent over the network. Next, if we didn't explicitly specify a partition, the data is sent to a partitioner. The partitioner will choose a partition for us, usually based on the `ProducerRecord` key. Once a partition is selected, the producer knows which topic and partition the record will go to. It then adds the record to a batch of records that will also be sent to the same topic and partition. A separate thread is responsible for sending those batches of records to the appropriate Kafka brokers. Source: Kafka: The Definitive Guide (2nd Ed.), O'Reilly Publishing (2021).

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 ProducerRecord Topic [Partition] [Key] Value Send () Serializer Partitioner When successful, return metadata If can't retry, throw exception Retry? Yes Fail? Yes Topic A Partition 0 Batch 0 Batch 1 Batch 2 Topic B Partition 1 Batch 0 Batch 1 Batch 2 Kafka broker *Figure 3-1. High-level overview of Kafka producer components* Source: Kafka: The Definitive Guide (2nd Ed.), O'Reilly Publishing (2021). Kafka Producer records include information identifying a specific topic.

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**

	 Constructor Summary
	Constructors
	Constructor and Description
	`ProducerRecord(String topic, Integer partition, K key, V value)`
	Creates a record to be sent to a specified topic and partition
	`ProducerRecord(String topic, Integer partition, Long timestamp, K key, V value)`
	Creates a record with a specified timestamp to be sent to a specified topic and partition
	`ProducerRecord(String topic, K key, V value)`
	Create a record to be sent to Kafka
	`ProducerRecord(String topic, V value)`
	Create a record with no key
	
	Source:
	https://kafka.apache.org/0102/javadoc/org/apache/kafka/clients/producer/ProducerRecord.html .

<table>
<tr><th colspan="2">U.S. Patent No. 8,407,722 (Claim 14)</th></tr>
<tr><th>Claim(s)</th><th>Example American Count 2 Systems and Services</th></tr>
<tr><td></td><td>

Method Detail

partition

```
int partition(java.lang.String topic,
              java.lang.Object key,
              byte[] keyBytes,
              java.lang.Object value,
              byte[] valueBytes,
              Cluster cluster)
```

Compute the partition for the given record.

Parameters:

`topic - The topic name`

`key - The key to partition on (or null if no key)`

`keyBytes - The serialized key to partition on( or null if no key)`

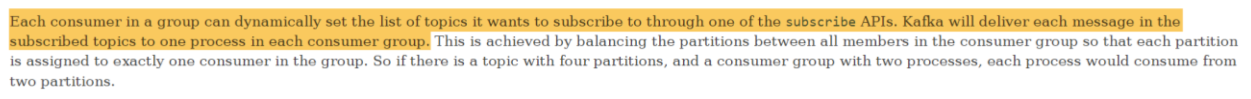
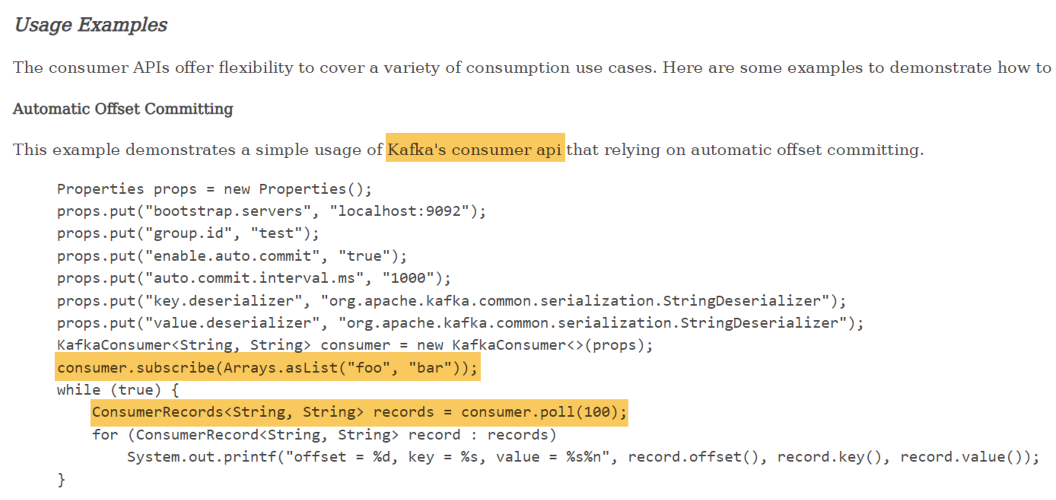
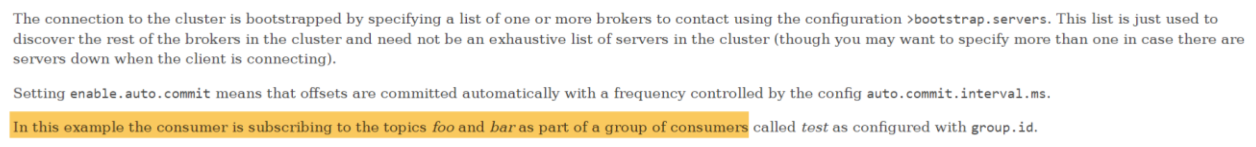
`value - The value to partition on or null`

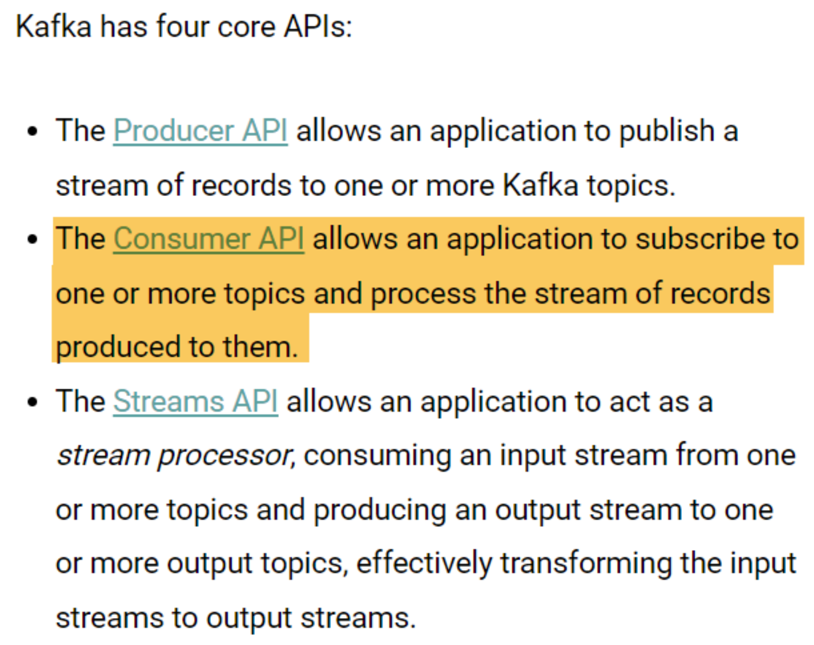
`valueBytes - The serialized value to partition on or null`

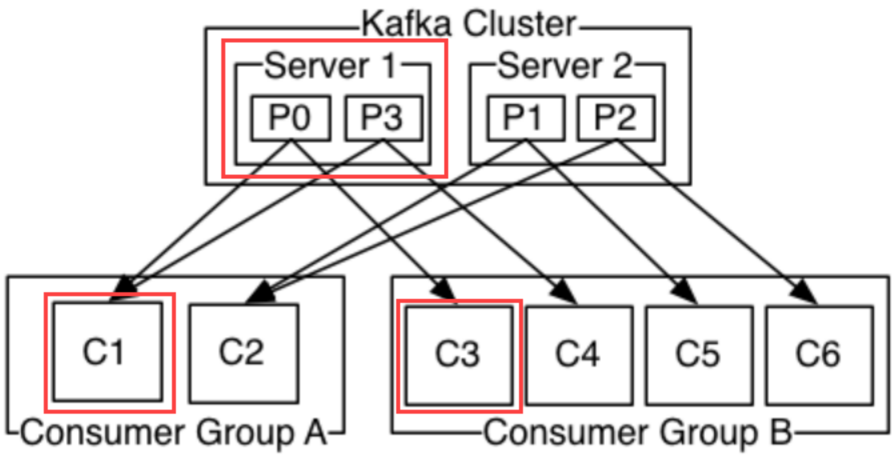
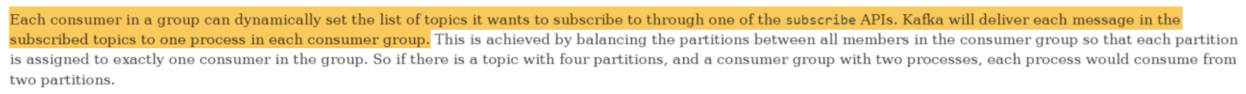
`cluster - The current cluster metadata`

Source: https://docs.confluent.io/5.4.11/clients/javadocs/org/apache/kafka/clients/producer/Partitioner.html.</td></tr>
</table>

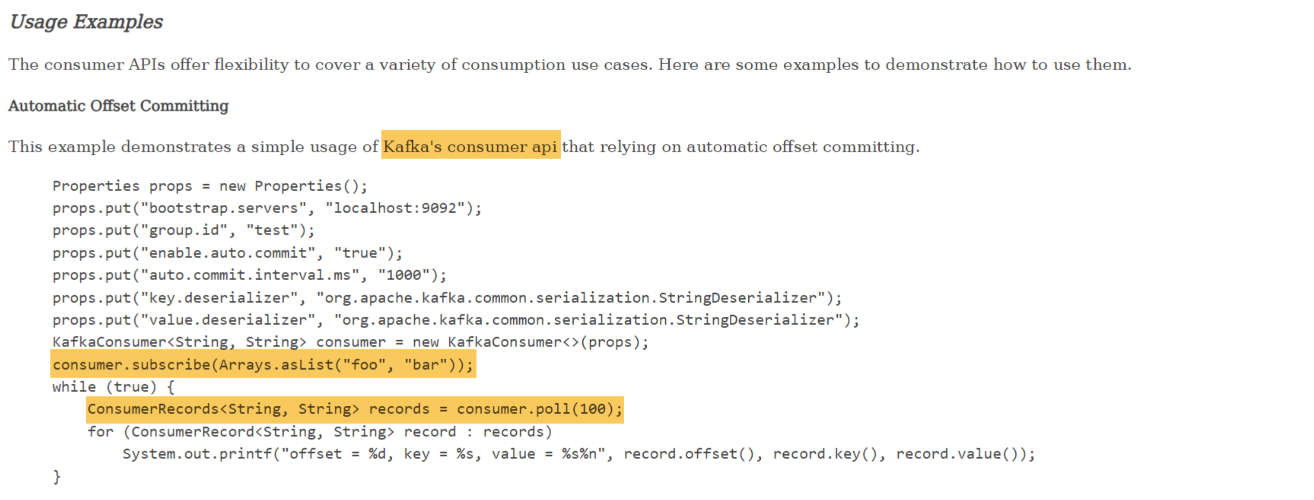
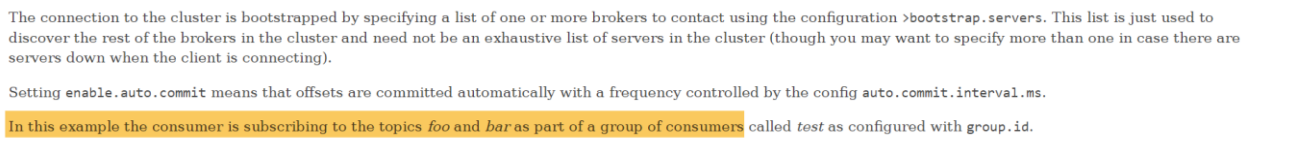
U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**

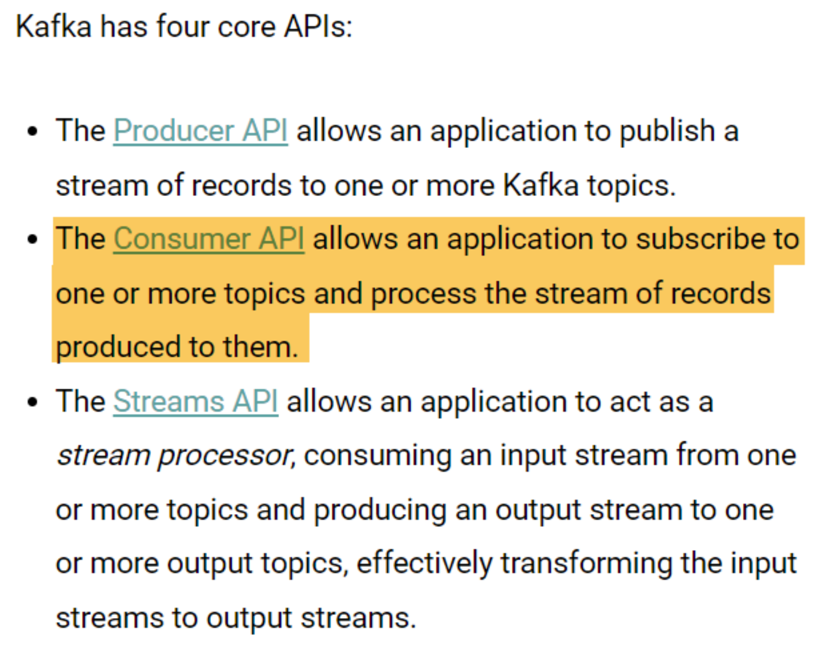
14[d] wherein the node is configured to identify the client device as a registered device and to route the update message to the client device, and	On information and belief, the American Count 2 Systems and Services practice the node is configured to identify the client device as a registered device and to route the update message to the client device.
	
	Kafka Consumers can set a list of topics that it wants to subcribe through subcribe APIs. Kafka delivers messages in a topic to those consumers that are subscribed to the topic.
	
	
	Each consumer in a group can dynamically set the list of topics it wants to subscribe to through one of the `subscribe` APIs. Kafka will deliver each message in the subscribed topics to one process in each consumer group. This is achieved by balancing the partitions between all members in the consumer group so that each partition is assigned to exactly one consumer in the group. So if there is a topic with four partitions, and a consumer group with two processes, each process would consume from two partitions.
	
	Source: https://kafka.apache.org/10/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html.
	
	
	Usage Examples
	The consumer APIs offer flexibility to cover a variety of consumption use cases. Here are some examples to demonstrate how to use them.
	Automatic Offset Committing
	This example demonstrates a simple usage of Kafka's consumer api that relying on automatic offset committing.
	`Properties props = new Properties();`
	`props.put("bootstrap.servers", "localhost:9092");`
	`props.put("group.id", "test");`
	`props.put("enable.auto.commit", "true");`
	`props.put("auto.commit.interval.ms", "1000");`
	`props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");`
	`props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");`
	`KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);`
	`consumer.subscribe(Arrays.asList("foo", "bar"));`
	`while (true) {`
	`    ConsumerRecords<String, String> records = consumer.poll(100);`
	`    for (ConsumerRecord<String, String> record : records)`
	`        System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(), record.value());`
	`}`
	
	Source: https://kafka.apache.org/10/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html.
	
	
	The connection to the cluster is bootstrapped by specifying a list of one or more brokers to contact using the configuration `>bootstrap.servers`. This list is just used to discover the rest of the brokers in the cluster and need not be an exhaustive list of servers in the cluster (though you may want to specify more than one in case there are servers down when the client is connecting).
	Setting `enable.auto.commit` means that offsets are committed automatically with a frequency controlled by the config `auto.commit.interval.ms`.
	In this example the consumer is subscribing to the topics *foo* and *bar* as part of a group of consumers called *test* as configured with `group.id`.
	
	Source: https://kafka.apache.org/10/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html.

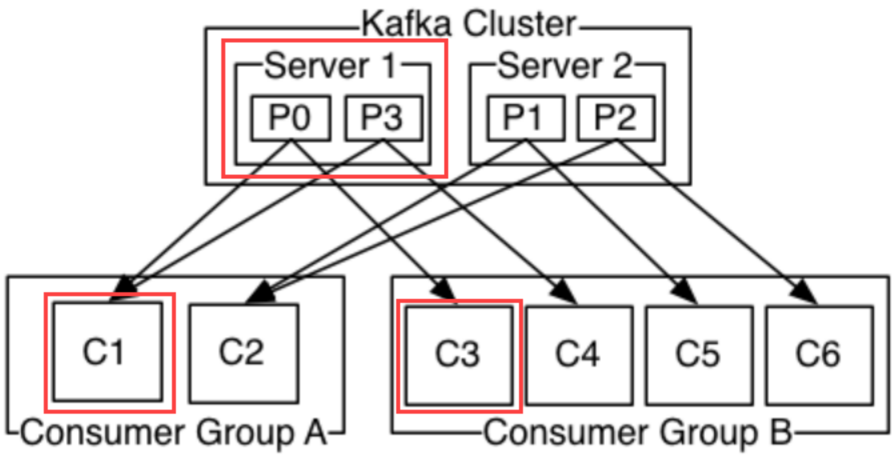
U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 Kafka has four core APIs: • The Producer API allows an application to publish a stream of records to one or more Kafka topics. • The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them. • The Streams API allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams. Source: https://kafka.apache.org/20/documentation.html.

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 Kafka Cluster Server 1 P0 P3 Server 2 P1 P2 C1 C2 C3 C4 C5 C6 Consumer Group A Consumer Group B Source: https://kafka.apache.org/20/documentation.html.
14[e] wherein the client device processes the update message upon receipt to update the property of the live object at the client device.	On information and belief, the American Count 2 Systems and Services practice the client device processes the update message upon receipt to update the property of the live object at the client device. Kafka Consumers can set a list of topics that it wants to subcribe through subcribe APIs. Kafka delivers messages in a topic to those consumers that are subscribed to the topic. Kafka Consumers process messages that are received via a Kafka broker.  Each consumer in a group can dynamically set the list of topics it wants to subscribe to through one of the `subscribe` APIs. Kafka will deliver each message in the subscribed topics to one process in each consumer group. This is achieved by balancing the partitions between all members in the consumer group so that each partition is assigned to exactly one consumer in the group. So if there is a topic with four partitions, and a consumer group with two processes, each process would consume from two partitions. Source: https://kafka.apache.org/10/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html.

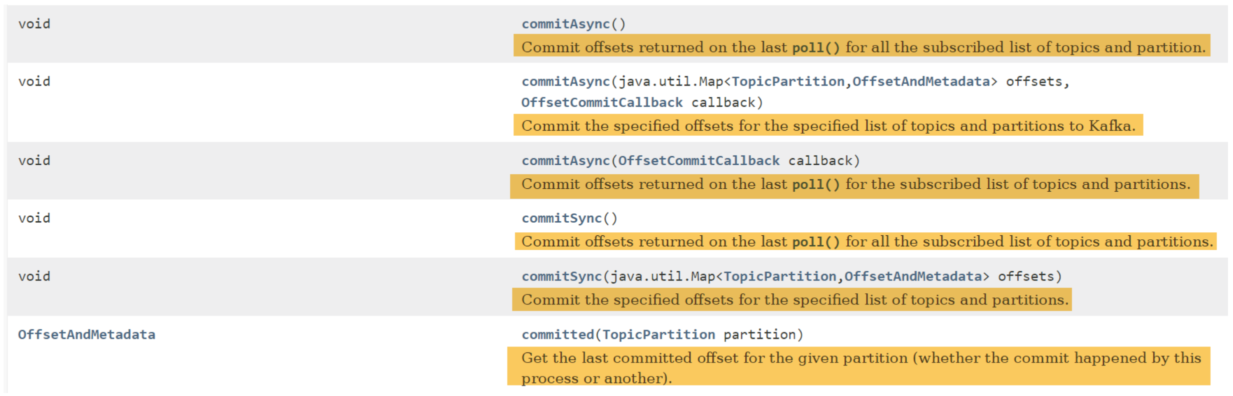
U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**

	 *Usage Examples*
	The consumer APIs offer flexibility to cover a variety of consumption use cases. Here are some examples to demonstrate how to use them.
	Automatic Offset Committing
	This example demonstrates a simple usage of Kafka's consumer api that relying on automatic offset committing.
	`Properties props = new Properties();`
	`props.put("bootstrap.servers", "localhost:9092");`
	`props.put("group.id", "test");`
	`props.put("enable.auto.commit", "true");`
	`props.put("auto.commit.interval.ms", "1000");`
	`props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");`
	`props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");`
	`KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);`
	`consumer.subscribe(Arrays.asList("foo", "bar"));`
	`while (true) {`
	`    ConsumerRecords<String, String> records = consumer.poll(100);`
	`    for (ConsumerRecord<String, String> record : records)`
	`        System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(), record.value());`
	`}`
	
	Source: https://kafka.apache.org/10/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html.
	
	 The connection to the cluster is bootstrapped by specifying a list of one or more brokers to contact using the configuration >`bootstrap.servers`. This list is just used to discover the rest of the brokers in the cluster and need not be an exhaustive list of servers in the cluster (though you may want to specify more than one in case there are servers down when the client is connecting).
	Setting `enable.auto.commit` means that offsets are committed automatically with a frequency controlled by the config `auto.commit.interval.ms`.
	In this example the consumer is subscribing to the topics *foo* and *bar* as part of a group of consumers called *test* as configured with `group.id`.
	
	Source: https://kafka.apache.org/10/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html.

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 Kafka has four core APIs: • The Producer API allows an application to publish a stream of records to one or more Kafka topics. • The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them. • The Streams API allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams. Source: https://kafka.apache.org/20/documentation.html.

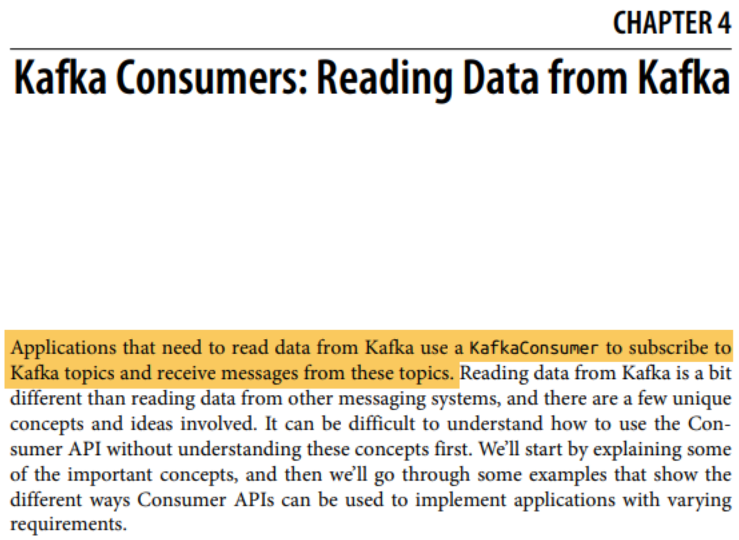
U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	

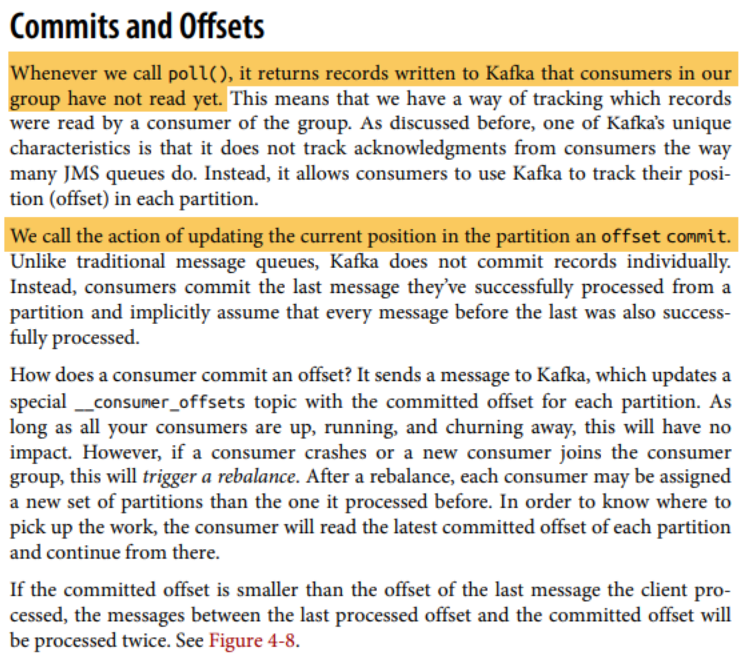
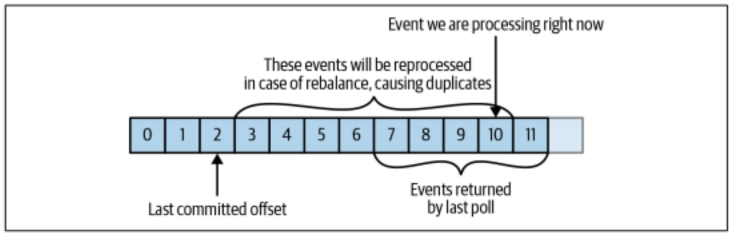
Source: https://kafka.apache.org/20/documentation.html.



void	commitAsync() Commit offsets returned on the last **poll()** for all the subscribed list of topics and partition.
void	commitAsync(java.util.Map<TopicPartition,OffsetAndMetadata> offsets, OffsetCommitCallback callback) Commit the specified offsets for the specified list of topics and partitions to Kafka.
void	commitAsync(OffsetCommitCallback callback) Commit offsets returned on the last **poll()** for the subscribed list of topics and partitions.
void	commitSync() Commit offsets returned on the last **poll()** for all the subscribed list of topics and partitions.
void	commitSync(java.util.Map<TopicPartition,OffsetAndMetadata> offsets) Commit the specified offsets for the specified list of topics and partitions.
OffsetAndMetadata	committed(TopicPartition partition) Get the last committed offset for the given partition (whether the commit happened by this process or another).

Source: https://kafka.apache.org/10/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html.

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 CHAPTER 4 Kafka Consumers: Reading Data from Kafka. Applications that need to read data from Kafka use a `KafkaConsumer` to subscribe to Kafka topics and receive messages from these topics. Reading data from Kafka is a bit different than reading data from other messaging systems, and there are a few unique concepts and ideas involved. It can be difficult to understand how to use the Consumer API without understanding these concepts first. We'll start by explaining some of the important concepts, and then we'll go through some examples that show the different ways Consumer APIs can be used to implement applications with varying requirements. Source: Kafka: The Definitive Guide (2nd Ed.), O'Reilly Publishing (2021).

U.S. Patent No. 8,407,722 (Claim 14)	
Claim(s)	**Example American Count 2 Systems and Services**
	 **Commits and Offsets** Whenever we call `poll()`, it returns records written to Kafka that consumers in our group have not read yet. This means that we have a way of tracking which records were read by a consumer of the group. As discussed before, one of Kafka's unique characteristics is that it does not track acknowledgments from consumers the way many JMS queues do. Instead, it allows consumers to use Kafka to track their position (offset) in each partition. We call the action of updating the current position in the partition an `offset commit`. Unlike traditional message queues, Kafka does not commit records individually. Instead, consumers commit the last message they've successfully processed from a partition and implicitly assume that every message before the last was also successfully processed. How does a consumer commit an offset? It sends a message to Kafka, which updates a special `__consumer_offsets` topic with the committed offset for each partition. As long as all your consumers are up, running, and churning away, this will have no impact. However, if a consumer crashes or a new consumer joins the consumer group, this will *trigger a rebalance*. After a rebalance, each consumer may be assigned a new set of partitions than the one it processed before. In order to know where to pick up the work, the consumer will read the latest committed offset of each partition and continue from there. If the committed offset is smaller than the offset of the last message the client processed, the messages between the last processed offset and the committed offset will be processed twice. See Figure 4-8. Source: Kafka: The Definitive Guide (2nd Ed.), O'Reilly Publishing (2021).  Event we are processing right now These events will be reprocessed in case of rebalance, causing duplicates 0 1 2 3 4 5 6 7 8 9 10 11 Last committed offset Events returned by last poll *Figure 4-8. Reprocessed messages* Source: Kafka: The Definitive Guide (2nd Ed.), O'Reilly Publishing (2021).